Driving Requirements Evolution by Engineers' Opinions

Kyanna Dagenais
McMaster University
Hamilton, Canada
dagenaik@mcmaster.ca

Istvan David
McMaster University
Hamilton, Canada
istvan.david@mcmaster.ca

ABSTRACT

Requirements are often incomplete or imprecise. Especially when innovation is a pronounced aspect of product development, sufficiently refined requirements can only be obtained by leveraging engineering knowledge gained through the exploration of innovative designs. However, such innovative designs often contradict prevalent requirements and might be deemed incorrect unless requirements evolve. In this paper, we present a method to drive requirements evolution by engineering opinions—early indicators of emergent engineering knowledge. Opinions about the suitability of a new design emerge earlier than hard evidence can be produced, potentially accelerating the evolution of requirements and saving time and costs. In this work, we develop a sound formal framework to inform requirements engineers about the potential need for requirements evolution based on engineering opinions. We formalize engineering opinions in terms of subjective logic and unify them with key concepts of model-driven engineering.

CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**; **Uncertainty quantification**.

KEYWORDS

collaboration, consistency, design space exploration, model-driven engineering, ontologies, parallel engineering, uncertainty

ACM Reference Format:
Kyanna Dagenais and Istvan David. 2024. Driving Requirements Evolution by Engineers' Opinions. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3652620.3688566

1 INTRODUCTION

Proper elicitation, formalization, and interpretation of requirements are key to developing correct software and systems [7]. However, requirements tend to be incomplete, imprecise, and misleadingly incorrect [22], leading to costly and failing projects. Especially in domains that require continuous innovation, eventual requirements are often shaped by engineers and experts. For example, in automotive software systems engineering, it is common to have high-level requirements to outline desired features by reusing ideas from previous projects [2], and allow detailed specifications to emerge at the leaves of the system decomposition tree [60]. This is due to the high degree of uncertainty [54] on the customer's side in complex engineering projects, caused by factors such as missing information or expertise, and the client stakeholders' difficulties in separating requirements from desires [24]. In contrast, important insights are gained on the engineering side by exploring the design space [2] and encountering promising alternative designs.

A large portion of alternative designs might be incompatible with the requirements, but engineers may still be confident in their design choices. Such cases are the ones when innovative ideas emerge and drive the evolution of requirements. Unfortunately, verifying the correctness of alternative designs might be cumbersome (e.g., a physical prototype might be required to be built and tested in a wind tunnel), slowing down the engineering and innovation process. To foster agility in engineering processes, we argue that engineers should be to able express their **opinions** about the worthwhileness of an alternative design to drive requirements toward beneficial directions. However, working with inherently subjective constructs such as opinions should still happen in a rigorous and formal fashion to remain systematic in engineering processes.

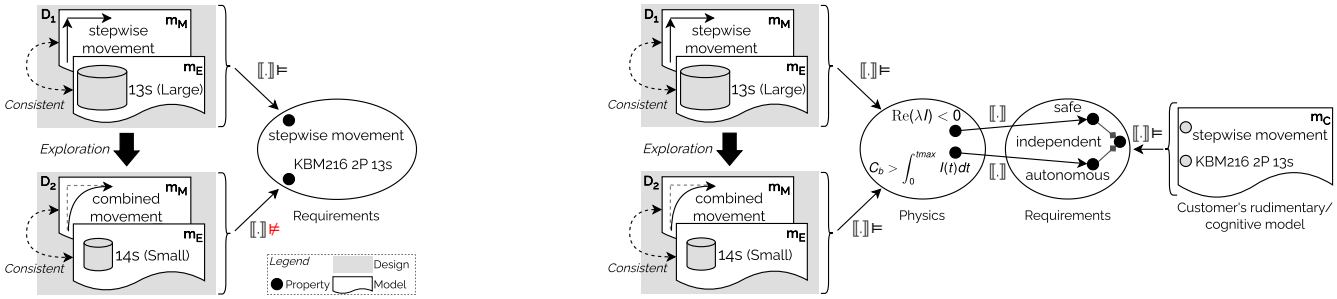
In this paper, we present a method to drive the evolution of requirements from the engineering side, by considering subjective opinions of experts (e.g., engineers, researchers, domain experts). Our goal is to notify requirements engineers as early as possible about the emergence of alternative designs that are **likely** worth presenting to clients for validation. To this end, we construct a formal framework with safety and liveness guarantees, based on subjective logic [33]. Subjective logic promotes opinions—e.g., engineers' confidence in their design—to first-class citizens, and allows for sound conclusions despite the subjective aspects it codifies.

Unfortunately, the complex framework of subjective logic limits its applicability. Most techniques that rely on it resort to simplifications, e.g., assuming that humans can express their opinions in terms of mathematical abstractions [8]; or emulating human opinion by objectively measured metrics, taking away the core subjective element of the approach [59]. Aptly, however, the formal facilities of model-driven engineering (MDE) [52] allow for a refined setup and calibration of subjective logic through a blend of objective factors, such as model correctness and consistency; and subjective factors, such as the heuristics engineers have about model correctness through perceived consistency [17]. We develop our method accordingly to avoid simplification of subjective logic and leverage it to its fullest extent.

The theory developed in this work has been successfully applied in our recent work on opinion-guided reinforcement learning [13] and forms the foundation for our work in reinforcement learning-driven model repair [12].

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688566



(a) Implicit semantics lead to perceived incorrectness **(b) Explicitly modeled semantics allow for sound conclusions about correctness**

Figure 1: Running example: requirements and engineering models of an automated guided vehicle

Contributions. The contributions of this paper are the following.

- We define a **method to incorporate engineering opinions** into the improvement of requirements.
- We define a **systematic approach** to **employing subjective logic** by MDE concepts.
- We identify **challenges and research directions** for the model-driven engineering community.

2 DEMONSTRATIVE CASE

To illustrate our points, we rely on a real industry case of developing an industrial automated guided vehicle (AGV). AGVs are portable robots, often implemented as line followers [53], widely used in industry settings, especially to carry payloads between locations.

The engineers are tasked with designing an AGV that is (i) safe and (ii) autonomous. The client is represented by a technical stakeholder who articulates the requirements based on a previous project.

Safety is associated with the stability of the AGV, i.e., preventing tilting and toppling. Based on a previous project, the client suggests ensuring safety by not allowing the AGV to move ahead and change direction simultaneously. That is, the movement must be sequential.

Autonomy is defined as the ability to carry out a mission without human intervention. In this electrified vehicle, autonomy mostly comes down to the selection of the battery to ensure sufficient charge. Based on a previous project, the client suggests choosing a specific battery from the supplier's catalog.

Independence is the freedom to carry out missions upon meeting safety and autonomy requirements. That is, the AGV must be both *safe* and *autonomous* to be considered *independent*.

Safety and autonomy can be verified through simulations. Verifying independence requires developing a prototype and passing customer-defined test scenarios. However, engineers, from experience, can tell that safety and autonomy *usually* imply independence.

2.1 Engineering process

2.1.1 Requirements. The set of requirements the client has articulated is shown in Fig. 1a. Safety requires *stepwise movement*; and autonomy requires selecting the *Kokam KBM216 2P 13S* battery.

2.1.2 Initial design. Design D_1 consists of the mechanical model (m_M) and the electrical model (m_E). Following the recommendations of the technical stakeholder, the mechanical engineer implements a *stepwise movement* mechanism that either moves the AGV forward or changes direction (but not at the same time); and the electrical engineer chooses the *13S* battery. We see that design D_1 in Fig. 1a satisfies the required properties, i.e., the required movement mechanism has been implemented, and the required battery has been selected. Therefore, the design is considered to be *correct*.

2.1.3 Design space exploration. Suspecting the existence of more optimal designs, the engineers explore the design space and stumble upon alternative design D_2, shown at the bottom of Fig. 1a. After simulations, they realize that *combined movement* that allows the AGV to move forward and change direction simultaneously will not lead to instability, i.e., it is safe. By choosing this movement mechanism, the length of the AGV's trajectory decreases by about 20% in such road segments. The engineers estimate that about half of the road segments would utilize a combined movement, shortening mission distances by 10%. This is a substantial difference and allows for an alternative, smaller battery, the *14S*. This design is now lighter and, consequently, more energy-efficient than the original.

2.1.4 Engineering intuition vs. verified design. Engineers verify the *safe* and *autonomous* properties through simulation. However, verifying *independence* is not feasible in a rapid exploration phase, as it requires developing a physical prototype. Thus, the engineers have no proof that the encountered alternative design is correct. Still, they judge that (i) the likelihood that their design is correct is high and (ii) the benefits of the design are sufficient to merit an investigation of requirements. Seemingly, the engineers did not change many parameters in the design either, i.e., they did not venture far in the design space, which increases the certainty of their opinions.

2.1.5 Incorrect design? Unfortunately, design D_2 in Fig. 1a is not what the client required. The design now, seemingly, does not satisfy the required properties ($[\![.]\!] \not\models$), i.e., it is deemed *incorrect*.

As shown in Fig. 1b, what we treated as requirements previously are essentially the customer's rudimentary, cognitive model m_C of the envisioned system, chosen in a way that they satisfy the actual set of required properties P_{req}. As they interpret the requirements, the engineers map the required properties into a more appropriate semantic domain, in this case, ordinary physics. The mechanical engineer interprets the safety requirement in terms of Newtonian physics, specifically in relation to the stability of the AGV; and in this view, stability requires the real part of the eigenvalue of the inertia matrix to be negative, i.e., $\mathrm{Re}\{\lambda I\} < 0$. Similarly, the electrical

engineer understands the autonomy property as the capacitance of the battery being greater than the current drawn by the AGV during its mission, i.e., $C_b > \int_0^{tmax} I(t)dt$. Such interpretations through the preferred semantic domain might be a natural way to look at the requirements or an artifact of a constructive argument. For example, the mechanical engineer might introduce the notion of inertia into the definition of safety when exploring more complex movement patterns, such as *combined movement*. With the notion of inertia being explicit, the engineer can now reason about velocity and arrive at a design in which combined movement at a lower velocity still satisfies safety constraints.

2.2 Lessons learned and requirements

First, Sec. 2.1.4 demonstrated the benefits of **a lightweight and rapid approach to channeling engineers' insights** into requirements management. Gathering hard evidence about the appropriateness of alternative designs might not always be feasible or might be cumbersome. But engineers might have **well-informed opinions** that carry immense value for requirements engineers already in the early phases of design [16]. For example, opinions as early indicators of emerging new knowledge, could trigger the evolution of requirements. Further safeguards to considering engineers' opinions might be desired, e.g., factoring in **how far the engineers ventured into the design space** during exploration. Obviously, the more parameters the engineers change in the alternative design, the less certain their judgment might be.

Second, Sec. 2.1.5 demonstrated that **maintaining an ontological view** allows for a more precise definition of *correctness*. An incorrect design might be an artifact of inappropriately formulated or partially elicited requirements. Thus, we need clearly expressed semantics to allow for sound notions of correctness and consistency.

By taking an ontological stance, subjective opinions can be informed through sound concepts. In this paper, we develop a formal framework to support such ambitions. Next, we provide a brief overview of the formal underpinnings of our framework.

3 BACKGROUND: SUBJECTIVE LOGIC

To formalize engineers' opinions, we rely on subjective logic.

Subjective logic [33] is an extension of probabilistic logic [1], in which users can express subjective opinions by quantified parameters of belief and certainty. Opinions are formed from a belief component and an uncertainty component. In statistics and economics, the uncertainty component of subjective logic is often called second-order probability [25], and is represented in terms of a probability density function over first-order probabilities.

Quantifying subjective opinions is an improvement over traditional logic as it systematically promotes opinions to first-class citizens. Informed opinions can be early indicators of emerging knowledge and often, opinions are the only available insights in engineering. For example, in Sec. 2, verifying the correctness of the overall design is not feasible, but the joint opinion of engineers about the utility of the design drives the evolution of requirements.

Formal underpinnings. Given a boolean predicate x, a binomial opinion regarding the truth of x is given by $\omega_x = (b_x, d_x, u_x, a_x)$, where b_x represents the belief in the truthfulness of x; d_x represents the disbelief in the truthfulness of x; u_x quantifies the degree of epistemic uncertainty or uncommitted belief concerning x; and a_x represents the base rate, i.e., the prior probability of x in the absence of (dis)belief. For each parameter, $0 \leq b_x, d_x, u_x, a_x \leq 1$. These parameters satisfy the following conditions.

$$b_x + d_x + u_x = 1 \tag{1}$$

$$P_x = b_x + a_x u_x. \tag{2}$$

Equation 2 expresses the *projected probability* of an opinion, effectively transforming subjective opinions into the probability domain, where $P(x)$ is the probability that boolean predicate x holds. It is clear that as uncertainty u_x increases, projected probability P_x is closer to base rate a_x. In contrast, as uncertainty u_x decreases, projected probability P_x is closer to that of the belief parameter b_x. Both Equations 1 and 2 are important invariants that we rely on throughout the paper, particularly in Sections 4 and 5.

Example. The opinion of the mechanical engineer E_M about the appropriateness of the design could be formulated as $\omega_x = (0.8, 0.0, 0.2, 0.5)$, meaning the engineer has a high belief ($b_x = 0.8$) that the design should be considered; albeit with some uncertainty ($u_x = 0.2$); and since there is no evidence to decide whether the design useful or not, the base rate is set to $a_x = 0.5$. By Equation 2, this opinion translates to a projected probability of $P_x = 0.9$. This probabilistic value, in turn, is more intuitive for a requirements engineer who needs to decide whether or not to trigger the evolution of requirements based on the engineer's opinion.

Fusing opinions. Informed opinions work well at scale. That is, the more experts that express their opinions, the higher the credibility of a collective opinion. Collective opinions can be inferred by semantically sound fusion operators [35]. This is yet another improvement of subjective logic over traditional logic and probability calculus, as these formalisms are not designed to handle different beliefs about the same statement. A fusion operator is a function $f : \Omega \times \Omega \rightarrow \Omega$ that maps a pair of subjective opinions onto a new, joint subjective opinion. The right operator must be chosen based on its fit for purpose. A detailed account of fusion operators is given in Sec. 4.4.

Challenges in employing subjective logic. The high expressive power comes with added challenges in employing subjective logic, particularly in controlling its parameters. There are two schools of thought to address this problem. In permissive techniques, setting uncertainty and belief-disbelief are deferred to the user [8, 32], and it is assumed that these parameters will become available at some point. It is, however, usually not explained when, how, and who has to set these parameters. Navarrete and Vallecillo [46] have users provide their opinion as a prior probability, a posterior probability, and an uncertainty. Belief and disbelief are derived from the posterior probability via a mapping from traditional probability as defined in [33]. In restrictive techniques, users are not asked to parameterize the framework, and rather, parameters are controlled by external measures [34, 43]. Unfortunately, this takes away subjective elements. Margoni and Walkinshaw [43] and Walkinshaw and Shepperd [59] use statistical evidence to set values, employing subjective logic to analyze empirical studies and experimental results, respectively. To represent confidence in assurance cases, Duan et al. [19] posit the use of Baconian probability to simplify the method of determining uncertainty. Likewise, Jøsang and Bondi [34] suggest

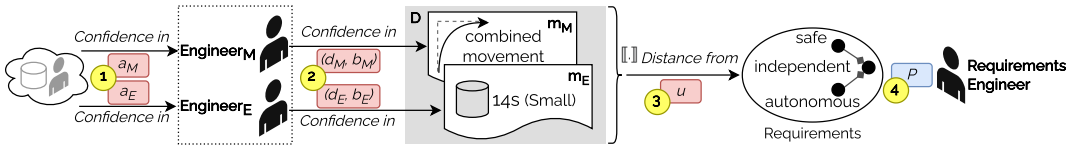


Figure 2: Overview of the approach with the main steps (1–4) and information highlighted: a – base rate (confidence in engineers); d and b – disbelief and belief (engineers' confidence in the design); u – uncertainty of engineers' decisions; P – probability that requirements need to evolve

using statistical evidence; as well as the use of a questionnaire to aid users in determining values. Similarly, Cyra and Górski [11] create an assessment scale that maps to predetermined values for binomial opinions. In regards to base rate a_x, some sources opt to set this value via statistical evidence [34]. Navarrete and Vallecillo [46] allow users to directly indicate the base rate, and Walkinshaw and Shepperd [59] assume $a_x = 0.5$. Evidently, configuring the framework of subjective logic is not a trivial task, and systematic methods are lacking. Our work links parameters of subjective logic to well-established MDE concepts to aid their configuration.

4 APPROACH

In this section, we present our approach to driving requirements evolution by expert opinions, and illustrate it through the case in Sec. 2. To formalize opinions, we leverage subjective logic [33]. As discussed in Sec. 3, the increased descriptive power of subjective logic comes at the expense of increased complexity in configuring its parameters. To overcome this issue, we develop a systematic approach in which we drive subjective logic by mapping notable properties of model consistency and correctness [17] onto the parameters of subjective logic as shown in Tab. 1 and Fig. 2.

Overview. At the end of the process in Fig. 2, the requirements engineer needs to be presented with a probability regarding the need for requirements evolution. Achieving this requires four steps.

In step 1, the confidence in engineers' ability to come up with correct and useful designs is defined. This information captures the perceived ability of the specific engineer in a class of tasks, e.g., the ability to judge mechanical designs. We use the base rate parameter a of subjective logic to encode this information. It is contextual information, as it is determined by the company's or team's context, rather than an specific problem at hand. As such, this parameter should be controlled by someone who can judge the performance of the engineers, such as a project manager or team lead.

In step 2, engineers express their confidence in the correctness and ability of their design to drive requirements evolution. We use the (dis)belief parameters d, b of subjective logic to capture this information. This is subjective information, as it depends entirely on the engineers' perception of the qualities of their work. As such, this parameter is controlled directly by the engineer.

In Step 3, the distance to requirements is calculated to capture the uncertainty of engineers' decisions. This metric acknowledges that the farther engineers deviate from the requirements during the design space exploration process, the less certain their judgment might become. In essence, the farther engineers venture into the design space, the higher the uncertainty of their beliefs. As explained in Sec. 3, this is the second-order probability of subjective opinions, which is captured in the uncertainty parameter u of subjective logic. Uncertainty is objective information in our case, as it depends on a measurable metric. As such, this parameter is controlled automatically by our framework.

Table 1: Separation of concerns and their mapping onto SL

Concern type	Subjective logic parameter	Responsible stakeholder	Derived from
Contextual	base rate (a)	PM, team lead	Seniority, historical accuracy
Subjective	dis/belief (d, b)	Engineer	Consistency
Objective	un/certainty (u)	*automated*	Distance from base correctness

Finally, in Step 4, the probability of needing to evolve the requirements is presented to the requirements engineer. To obtain the projected probability of subjective opinions, we rely on Equation 2 and present the requirements engineer with the probabilistic value P. If this probability meets a pre-defined threshold $P > \tau$, the requirements engineer can infer that requirements might need to evolve, based on the opinions of engineers.

In the remainder of this section, we elaborate on how contextual (Sec. 4.1), objective (Sec. 4.2), and subjective (Sec. 4.3) components are derived, and fused into a single probability metric (Sec. 4.4).

4.1 Contextual component: confidence in engineers

In the first step of the approach (1), confidence in the engineers is calibrated. As shown in Tab. 1, this information is mapped onto base rate a, representing prior probability in the absence of (dis)belief.

This contextual information specifies the trust a company or manager has in the engineers. This information is quantified by the likelihood that the alternative design proposed by the specific engineer turns out to be correct. The base confidence in an engineer can be interpreted as the formal encoding of domain expertise attributed to seniority, or a track record of historical accuracy in driving requirements evolution. In most cases, there is no clear evidence to change the prior probability from the default $a = 0.5$. However, base rate can be calibrated in a more constructive way based on historical data and appropriate internal assessment procedures.

A good example of calibrating base rates comes from an earlier industry project in which we collaborated with one of the largest financial institutions in the country. Our goal was to promote "skill" to a first-class citizen in agile software engineering to allow reasoning about optimal allocations of tasks (with required skills) to engineers (with possessed skills). The skills of engineers were assessed regularly in a quarterly meeting by their team leader. Base

rate is analogous to this skill level. It can be calibrated manually (as done in our previous project) or in an automated fashion.

Example 1. Since we have no argument to decide otherwise, the base rates of both engineers should be calibrated at $a = 0.5$, representing the situation where no a-priori information is available about the engineers' backgrounds. However, for the sake of demonstration, we assume that electrical engineer E_E is a junior professional and mechanical engineer M_E is a senior with many years of experience at the company. Thus, we estimate their respective base rates at $a_{E_E} = 0.2$ and $a_{E_M} = 0.6$. This means that the electrical engineer is right in 20% of the cases when the appropriateness of an alternative design has to be judged. (Base rate should account for domain expertise and, ideally, should be set per skill group, as explained above. Furthermore, deviation from $a = 0.5$ should be based on clear and consistently repeatable assessments.)

4.2 Subjective component: engineers' confidence in design by consistency

In the second step of the approach (2), the engineers express their confidence in their design. As shown in Tab. 1, this information is mapped onto the disbelief d and belief b parameters of SL, representing the (dis)belief engineers have in the design at hand.

4.2.1 Consistency as a heuristic to correctness. For a formal definition of meeting requirements, i.e., having a correct design, we recall the related definition of David et al. [17, Definition 8]:

Definition 4.1 (Correctness of a design). Design D is said to be correct with respect to its set of required properties $P_{req}(D)$ iff $\forall p \in P_{req}(D) : [\![D]\!] \models p$, where $[\![.]\!]$ denotes semantic mapping.

Here, the set of required properties of design D is given as $P_{req}(D) = \bigcup_{m \in D} P_{req}(m)$, i.e., the required properties of the design is the set of required properties of its constituent models. In the running example, $P_{req}(D) =$ *safe*, *autonomous*, *independent*. The semantic mapping between *safe* and m_M is given by $[\![m_M]\!] \models (\mathrm{Re}\{\lambda I\} < 0)$, which can be evaluated through simulation.

When exploring alternative designs, it may not be feasible to prove correctness. Instead, engineers rely on their opinions about the correctness of design. As shown by David et al. [17], consistency is a heuristic to eventual correctness. That is, following the definition of Romanycia and Pelletier [50], consistency is a device, which «*one is not entirely confident will be useful*» in achieving correctness, but which «*one has a reason to believe will be useful*» in doing so. Thus, engineers can rely on the consistency of their models to increase their *belief* in the eventual correctness of the design.

Example 2. In the running example, the two engineers deviate from the correct baseline D_1 to find the more optimal D_2. However, as shown in Fig. 1b, each engineer has access to one property (*safety* and *autonomy*), and proving *independence* might not be feasible in their respective views. Instead, engineers rely on the presence of consistency as they both satisfy their required properties.

4.2.2 Consistency to formulate belief. We adopt the related definition of David et al. [17, Definition 10] as follows.

Definition 4.2 (Consistency of two models). Models m_i, m_j are said to be consistent w.r.t to property $p \in P$ iff $[\![m_i]\!] \models p \Leftrightarrow [\![m_j]\!] \models p$.

That is, consistency is concerned with whether models make the same (or congruent) statements about certain properties. In the trivial case, $[\![m_i]\!] \models p \wedge [\![m_j]\!] \models p$, i.e., both models satisfy property p, and thus, are deemed consistent. However, the definition allows for the non-trivial case too, in which $[\![m_i]\!] \not\models p \wedge [\![m_j]\!] \not\models p$, i.e., the two models are consistent but incorrect.

Def. 4.2 can be used in two ways to influence engineers' beliefs: in a binary way and in a quantified way. The rather trivial notion of binary inconsistency informs the engineer whether models are in a consistent state or not. Models m_i, m_j are inconsistent iff $\exists p \in P : ([\![m_i]\!] \models p \wedge [\![m_j]\!] \not\models p) \vee ([\![m_i]\!] \not\models p \wedge [\![m_j]\!] \models p)$. Binary inconsistency is not exactly useful in aiding engineers in formulating their belief in the correctness of their approach as consistent cases do not help determine just *how* confident the engineer should feel about the design. A more useful notion of inconsistency is quantified inconsistency [5], aiming to express the degree of inconsistency between two models. As opposed to the nominal level of measurement of binary inconsistency (named categories without order), quantified inconsistency is at least at the interval level of measurement (distance between values is meaningful). This richer semantic domain gives engineers a better grip on just how confident they should be about their design.

Example 3. The number of inconsistent properties is a good example of a simple consistency quantification metric.

$$h(D) = \left|\left(P_{sat}(m_i) \ominus P_{sat}(m_j)\right) \bigcap P'\right|, \tag{3}$$

where $\ominus$ is symmetric difference; $P' \equiv P_{req}(m_i) \bigcap P_{req}(m_j)$ is the overlap of concerns; and $P_{sat}(m) \subseteq P_{req}(m)$ are satisfied properties.

Calculating this distance assumes that there are (in)consistency management rules in place that allow for the efficient evaluation of some properties of interest that form the basis of h. Typically, a subset of ontological properties is available in the linguistic realm, i.e., the set of properties P in Equation 3 gets substituted by $P^{\mathcal{L}} \subseteq P$.

4.2.3 A language to express opinions. To express engineering opinions, an appropriate language is required. We use Likert-type rating scales to guide participants in expressing their answers. Likert-type scales are psychometric rating scales frequently employed in questionnaires to measure the attitude of participants towards a specific question [36]. Likert scales narrow the cognitive gap between human reasoning about confidence and the framework of subjective logic, and allow humans to express their opinions at a higher level of abstraction. A similar approach has been used by Jongeling and Vallecillo [32] to express opinions about model consistency resolution strategies. Following Vagias [55], we recommend the following Likert items to gauge confidence: {*Not confident at all*; *Slightly confident*; *Somewhat confident*; *Fairly confident*; *Completely confident*}. We then use an equidistant mapping of the Likert scale onto the disbelief-belief scale, assigning a b value proportionally to the order of the Likert item. (It is important to re-iterate here that mapping beliefs to a Likert-scale does not render them objective information.) Three example cases are shown in Tab. 2. However, the full mapping cannot be determined without fixing the uncertainty parameter u. In Sec. 4.3, we show how to deal with this last parameter and provide a full configuration of the SL framework.

Example 4. In the running example, the engineers explore alternative design D_2. They observe that w.r.t. the number of accessible properties (*safety* and *autonomy*), their inconsistency metric $h(D_2)$ evaluates to 0, giving them confidence in their design. Electrical engineer E_E has a simpler model and, thus, is *Completely confident* in the design, while the relatively more complex mechanical design makes mechanical engineer E_M slightly less confident, but still *Fairly confident* in the design. By Tab. 2, these choices map to $b_{E_E} = 1.0, d_{E_E} = 0.0$ and $b_{E_M} = 0.75, d_{E_M} = 0.25$. Adding these values to the choices of $a_{E_E} = 0.2$ and $a_{E_M} = 0.6$ in *Example 1*, the opinions of the engineers are now given by $\omega_{E_E} = (1.0, 0.0, u, 0.2)$ and $\omega_{E_M} = (0.75, 0.25, u, 0.6)$. However, without setting u, belief and disbelief values b and d are not precise. In the following, we derive the missing u value to fully configure ω_{E_E} and ω_{E_M}.

4.3 Objective component: engineers' certainty in their opinions by model distance

In the third step of the approach (3), the previous subjective opinions are augmented with an objective certainty metric, the distance of the design from the requirements. As shown in Tab. 1, this information is mapped onto the uncertainty u parameter of SL, representing the uncertainty that the engineers can properly judge their designs. As engineers deviate from the correct base model and leave an increasing number of required properties unsatisfied, we interpret their certainty as lower. Thus, we need a measure to assess the distance between the design and requirements.

4.3.1 Measuring distance. We rely on the following definition [14].

Definition 4.3. (Measure) A given set of values $\mathbb{M}$ with operations $0 :\rightarrow \mathbb{M}, + : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{M}$ (0 neutral, + associative and commutative) and an order relation $\leq$ on $\mathbb{M}$ is called a measure.

Def. 4.3 is a common definition of measure, and property counting as shown in Equation 3, meets this definition. Thus, to measure the distance between requirements and design, we rely on counting the ratio of satisfied and required properties in design D. More precisely, to map the distance between requirements and design onto the uncertainty value u of SL, we normalize it to the $0 \leq u \leq 1$ domain, where 0 represents situations when the design satisfies every required property, and 1 represents situations when the design does not meet any properties. The normalized distance of design D from the requirements is defined as $h^*(D) = 1 - \left(|P_{sat}(D)|/|P_{req}(D)|\right)$.

4.3.2 Full configuration of the SL framework. Now that the u parameter can be calibrated, we revisit Sec. 4.2 and obtain the full configuration of the subjective logic framework for our case. Given $0 \leq u \leq 1$, the remainder of opinion weights has to be distributed between b and d in accordance with Equation 1. The calculation method of belief b and disbelief d for the ith Likert item in an n-point Likert scale, in ascending order of confidence from least to most confident, are given by $b_i = i \times \frac{1-u}{n-1}$, and symmetrically, $d_i = (1-u) - b_i$. Three example mappings are shown in Tab. 2.

Example 5. In the running example, the alternative design D_2 has two proven required properties: *safety* and *autonomy*; but engineers cannot verify *independence* from their own domain-specific views, as proving this property requires building a prototype and testing it on customer-defined scenarios. Thus, the ratio of satisfied and required properties is $h^*(D_2) = \frac{2}{3}$, which is the u parameter in the SL framework underpinning our approach. Thus, $u = 0.667$. This case is shown in the third column of Tab. 2, resulting in the following mappings of Likert items to SL parameters: *Fairly confident* $\rightarrow b = 0.250$, and *Completely confident* $\rightarrow b = 0.333$. As seen here, belief and disbelief are discounted by uncertainty. The values in *Example 4* $b_{E_E} = 1.0, d_{E_E} = 0.0$ and $b_{E_M} = 0.75, d_{E_M} = 0.25$ are discounted to $b_{E_E} = 0.333, d_{E_E} = 0.0$ and $b_{E_M} = 0.250, d_{E_M} = 0.083$. The new values round out the running example as follows: $\omega_{E_E} = (0.333, 0.0, 0.667, 0.2)$ and $\omega_{E_M} = (0.250, 0.083, 0.667, 0.6)$.

Table 2: Example mappings of confidence levels onto the belief dimension of SL at different degrees of uncertainty u

Likert item	*u=0.0*		*u=0.2*		*u=0.667*	
	b	*d*	*b*	*d*	*b*	*d*
Not confident at all	0.00	1.00	0.0	0.8	0.000	0.333
Slightly confident	0.25	0.75	0.2	0.6	0.083	0.250
Somewhat confident	0.50	0.50	0.4	0.4	0.167	0.167
Fairly confident	0.75	0.25	0.6	0.2	0.250	0.083
Completely confident	1.00	0.00	0.8	0.0	0.333	0.000

4.4 Putting it all together: fusing belief and presenting probabilistic evidence

Finally, in the last step of the approach (4), all the previous contextual, subjective, and objective information is combined, translated into a probabilistic value $P = b + au$ (Equation 2), and presented to the requirements engineer. The engineer then compares this probabilistic value to a previously chosen threshold τ. In case of $P > \tau$, the requirements engineer decides to initiate the evolution of requirements; otherwise ($P \leq \tau$), evolution is rejected on account of low confidence in the alternative design.

Combined (joined) opinions are obtained through *fusing* single opinions. The seminal work of Jøsang [33] defines an array of fusion operators and classifies them according to situational characteristics to facilitate the selection of the most appropriate operator. Here, we rely on Belief Constraint Fusion (BCF), but other fusion operators can be used as well. BCF is an appropriate choice when agents have already made up their minds and will not seek compromise. Since the engineers formulate their opinions about the design independently from each other, it is fair to assume that engineers will, indeed, not seek compromise. Reasoning through domain-specific views also hinders seeking compromise.

A fused opinion $\omega^{\odot} = (b^{\odot}, d^{\odot}, u^{\odot}, a^{\odot})$ is calculated from the overlapping beliefs (called *harmony*) and non-overlapping beliefs (*conflict*) of individual opinions. $Harmony = b_1u_2 + b_2u_1 + b_1b_2$, which, given that in our approach $u = u_1 = u_2$, simplifies as follows.

$$Harmony = (b_1 + b_2)u + b_1b_2; \tag{4}$$

$$Conflict = b_1d_2 + b_2d_1. \tag{5}$$

The parameters of joint opinion $\omega^{\odot}$ are calculated as follows.

$$b^{\odot} = \frac{Harmony}{(1 - Conflict)};\ d^{\odot} = 1 - (b^{\odot} + u^{\odot}); \tag{6}$$

$$u^{\odot} = \frac{u_1u_2}{(1 - Conflict)} = \frac{u^2}{(1 - Conflict)}; \tag{7}$$

$$a^{\odot} = \frac{a_1(1-u_1) + a_2(1-u_2)}{2 - u_1 - u_2} = \frac{(a_1 + a_2)(1-u)}{2(1-u)} = \frac{a_1 + a_2}{2} \tag{8}$$

5 EVALUATION: (WHY) DOES THIS WORK?

Our approach works because it guarantees *safety* and *liveness* properties along the engineering endeavor. By Lamport [38], safety stipulates that "bad things" will not happen, and liveness stipulates that "good things" will eventually happen as the engineers collaborate. Such beneficial situations, coupled with the safety guarantees, eventually justify employing our approach.

5.1 Safety properties

5.1.1 Uncertain situations never overestimate confidence in engineers. It is expected that in uncertain situations, i.e., in case of vacuous opinions [33] when $u \approx 1$, the requirements engineer is not informed with a higher overall probability for change P than the base confidence in engineers. That is, $\lim_{u \to 1} P = \bar{a}$, where $\bar{a}$ denotes the average base rate assigned to the engineers.

The proof is trivial. From Equation 1, it follows that $u = 1 \Rightarrow b + d = 0$, that is $u = 1 \Rightarrow b = d = 0$. Substituting these into $P = b + au$ (Equation 2), we get $P_x(x|u = 1) = a$. That is, the requirements engineer will never be informed about a higher probability to trigger requirements evolution than their base confidence in an engineer to signal a true positive need for requirements change.

To generalize to more than one engineer, consider an aggregation function to obtain $\bar{a}$ from the set of base rates $A = \{a_n | n \in \mathbb{N}\}$. For example, BCF obtains the fused base rate by arithmetic mean (Sec. 4.4). This means, the requirements engineer is never informed about a higher probability of required evolution than the mean base confidence $\bar{a}$ in engineers.

5.1.2 Most incorrect designs are not proposed to the requirements engineer. This is a weak safety property as it does not hold for each case, but it holds for *reasonable* cases. As explained in Sec. 4, consistency of design D is a prerequisite to proposing it to the requirements engineer. Thus, inconsistent designs are surely not proposed. Unfortunately, consistency is a necessary *but not sufficient* requirement for correctness [17, Theorem 1]. This is a problem in situations when engineers have consistent models, but models are consistently *incorrect*, e.g., due to working under wrong assumptions or choosing the wrong frame of validity [45]. In such rare edge cases, engineers might have a high confidence in an incorrect design, and requirements engineers together with stakeholder clients must be able to identify such designs.

5.2 Liveness properties

5.2.1 Even novice experts can drive requirements evolution. The base rate is calibrated by assessing the experience of experts, approximating or deriving the likelihood of their input being valid and useful. Liveness, in this setup, means that for any base rate, even for a low one ($a \approx 0$), there exists a certainty-belief configuration that will meet an arbitrary threshold τ that will trigger the evolution of requirements. Formally, $\forall a : 0 \leq a \leq 1,\ \exists u : 0 \leq u \leq 1 : P > \tau$, where τ is the chosen threshold to initiate the requirements evolution process. The proof, again, is trivial, considering that $P = b + au$, from which, in the worst case of $a = 0$, it follows that $P_x(x|a = 0) = b_x$. Given that $1 = b + d + u$, it follows that $u + d < \tau \Rightarrow b > 1 - \tau \Rightarrow P > \tau$.

5.2.2 Requirements can evolve even at high acceptance thresholds. This means companies can opt for safe settings by choosing a comfortably high τ that will necessitate high belief and certainty to trigger the evolution of requirements; and gradually shift towards more permissive configurations with a lower τ. For proof, consider $\tau = 1$, i.e., requirements evolution requires a probability of 1. From $P = b + au$, it follows that $1 = b + au$; and considering Equation 1, it follows that $b + d + u = b + au$, simplifying to $d + u = au$. After organizing the equation, we get $\frac{d}{u} = a - 1$. Since $0 \leq d, u \leq 1$, it follows that $0 \leq \frac{d}{u}$, necessitating $0 \leq a - 1$, but due to $0 \leq a \leq 1$, it follows that $a - 1 = 0$, and $\frac{d}{u} = 0$ holds. Consequently, $d = 0$.

That is, meeting evolution trigger threshold $\tau = 1$ is still attainable with $a = 1$ and $d = 0$, i.e., with an expert whose input is always considered as long as they express no disbelief. As a corollary, it follows that the degree of certainty is irrelevant in this case. This is the profile of senior architects who often have an unquestioned role in the management of requirements and whose indication of inaccurate requirements is taken at face value.

6 DISCUSSION

We now discuss the key takeaways and outline some challenges and research opportunities for prospective researchers.

6.1 More agile requirements engineering

The most important takeaway of this paper is that **opinions can be effective driving factors** to requirements evolution. Opinions are formed more easily and faster than knowledge and hard evidence, and as demonstrated, when formally informed, engineering opinions can be of high utility in steering projects toward higher value-added results. Basing formal reasoning on opinions allows for leveraging early indications of new knowledge, and by channeling the intuitions of engineers into quantified metrics, evolutionary needs can be intercepted as early as possible.

The separation of concerns renders the decision whether to initiate evolution a truly **collaborative endeavor**, in which objective, subjective, and contextual information can be considered in their most appropriate form, thanks to subjective logic. The collective and collaborative setting around the engineering problem suggests a diverse set of viewpoints to render opinions relevant at scale.

Our approach fosters **more agile requirements management** by allowing engineers to explore the design space freely while working under proven safety guarantees (Sec. 5.1). The liveness properties of our approach (Sec. 5.2) suggest that promising results of exploration will be considered irrespective of the engineers' seniority or the risk-aversion of the company. These safety and liveness properties should motivate companies to adopt our approach.

Agile requirements management mechanisms, such as the one presented in this paper, are of particularly high value in settings without a properly codified problem space and new problem domains. When engineers have to address complex problems in new domains, **a shared cognitive model with the client stakeholders** is particularly challenging to establish. Extensive socialization [47]—e.g., working in a master-apprentice setup or observing practitioners—is a well-known technique to handle such cases, allowing for customers and engineers to tap into each other's expertise and chain of thought. However, socialization is not something

software and systems projects can typically afford. Our approach facilitates convergence towards shared cognitive models by basing its formal frameworks on opinions rather than hard evidence.

6.2 Systematic methods to subjective logic through sound MDE concepts

Subjective logic **promotes uncertainty** to a first-class citizen instead of forcing unnatural codification of truly human faculties and attitudes such as belief and opinions. Thus, subjective logic is an important improvement over logic frameworks typically encountered in MDE-related problems, such as first-order logic in consistency management [17], description logic in ontological reasoning [56], and modal logic in knowledge base evolution [57].

Unfortunately, the added modeling power comes at the cost of **particularly cumbersome deployment** and calibration of subjective logic. In Sec. 3, we elaborated on the restrictions and simplifications state-of-the-art approaches resort to in order to manage the complexity of subjective logic.

Our approach leverages the **full power of subjective** logic by separating its parameters into subjective, objective, and contextual concerns and establishing strong links with core MDE concepts, such as consistency and correctness. We recognize that controlling each parameter is not feasible for the end-user due to the immense cognitive load. Answering the simple questions of "what is your belief about X?" and "what is your certainty about your belief?" is confusing and challenging. Even the developers of subjective logic resort to higher-level DSLs to hide details of the framework and elevate the articulation of opinions to human levels [34, Sec. 5.2.1].

Subjective logic offers opportunities for **high levels of automation**. For example, engineers can be assisted in articulating their opinions by appropriate default belief-disbelief values through automated consistency assessment and reporting. A complete, easily accessible, and queryable history of engineer decisions would be a key feature to gauge the base rate of engineers in an objective way. History should be captured in a semi-formal fashion, e.g., using decision graphs [51] that are amenable to automated analysis.

6.3 Caveats and limitations

The main caveat of our approach is being limited to the development side (including engineers and stakeholders) and **not integrating client beliefs** into the reasoning process. We acknowledge the potential high benefits of modeling client beliefs in a truly end-to-end framework. However, assessing confidence in the client might pose challenges that are outside of an engineering company's reach. For example, asserting a base rate to client stakeholders might not be feasible. Furthermore, we are not aware of heuristics that could drive client belief in the systematic way model consistency drives engineers' beliefs.

Another caveat is related to the second, weak **safety guarantee** (Sec. 5). While *most* incorrect designs are not proposed to the requirements engineer, it is possible that some incorrect designs will be proposed. The inconsistency of the design will indicate sure inconsistency, and thus, we can safely handle such cases. However, consistent cases can still lead to incorrect designs. The safe handling of these situations requires human intelligence in requirements management and on the client side.

Finally, the **selection of the fusion** operator (Sec. 4.4) is a challenge in employing subjective logic [35]. However, in most cases, the differences between fusion operators is negligible compared to other factors in our approach, e.g., quantifying consistency and measuring the distance between design and requirements.

6.4 Challenges and research opportunities

Some notable challenges and opportunities for prospective researchers and developers are the following.

Efficient **semantic techniques and tools** are required to be researched in support of the outlined approach. Ontological reasoning has demonstrated benefits in heterogeneous settings in which stakeholders of disparate domains are involved, and common vocabularies are lacking [58]. The need for ontology-based requirements management techniques has been recognized before, e.g., in mechatronics [15]. However, to get there, integration of semantic techniques with the prevailing linguistic approaches is required [6]—but this remains an open challenge.

To allow for externalizing engineers' opinions, suitable **languages** are required. In our work, we resorted to Likert scales as they are designed to elicit human attitudes toward statements about phenomena of interest–opinions, basically. A similar technique was chosen by Jongeling and Vallecillo [32] in their subjective logic-based inconsistency management approach and by Jøsang and Bondi [34] in their approach for legal reasoning. We see opportunities in supporting our technique by domain-specific languages (DSL) [44], especially in blended fashion [4] with engineering modeling languages to further streamline the engineering process.

To accelerate the engineering and requirements evolution process, **automated derivation of meaningful belief defaults** could be developed. Our approach establishes a link between model consistency and belief, which could be further improved by informing the engineer about an approximate belief value. Alternatively, fully automated virtual experts could be developed who represent the fully synthetic belief in a project and could be treated as additional project members when it comes to driving requirements evolution.

On a related note, a **method and tooling for calibrating and maintaining the base rate** is required before our approach is deployed in real settings. In our experience from an industry project, regular individual assessments of engineers' base rates are a good first step that will establish methodological clarity. In the long run, however, proper automation and tooling are required to mitigate threats to validity in the assessment of human capabilities and skills. Structured design rationale techniques are prime candidates to provide the foundation for automated quantified assessment. Such directions have been explored by Salay et al. [51] who augment requirements models with rationale to reduce uncertainty. Design rationale has been used previously in other steps of the model-driven engineering process, e.g., to augment design decisions [48], goal models [42], and requirement models [26].

Finally, as a key software development challenge, efficient **end-to-end tool support** is required. A brief review of the seminal work of Hoffmann et al. [30] on the requirements for requirements management tools reveals that requirements engineering tools might provide appropriate foundations for the envisioned tool support,

but key features might be missing. Most notably, **quantified inconsistency reporting** could be developed to inform engineers about the degree of inconsistency in design and form a basis for informed opinions. As demonstrated in Sec. 4.2.2, quantified (in)consistency is an appropriate formal underpinning to formulating engineering opinions. Thus, lightweight mechanisms that maintain and report metrics of inconsistency are of particularly high utility. Such avenues have been explored both in the ontological [41], and linguistic realm [39], but appropriate tool support is yet to be developed.

7 RELATED WORK

Requirements in software and systems engineering are inherently uncertain and, some claim, are never complete [49]. To converge to sufficiently detailed requirements, systematic evolution mechanisms Ernst et al. [21], Joshi and Summers [37] are needed.

Uncertainty in requirements. Uncertainty is typically due to vague product strategy, missing key stakeholders, and missing or unavailable domain knowledge [20]. Salay et al. [51] promote uncertainty in requirements to a first-class citizen by using partial models. Partial models allow for accounting for missing or incomplete information in requirements. Partiality is expressed through four kinds of uncertainty annotations: the presence of a particular model element in a model; the number of model elements in the model; the distinctness of model elements; and the completeness of the model. As opposed to our approach, partial models focus on *structural* deficiencies of models; and the formalization of the modeler's belief requires a good understanding of the domain, e.g., of *what* may be missing from the model. Our approach, therefore, fosters a more streamlined articulation of engineers' belief and that, in a quantified fashion.

Involving stakeholders and triggering creativity to drive innovation. A particular need that contributes to uncertainty in requirements is the need for innovation during the engineering endeavor [2]. Much effort has been dedicated to involving *client stakeholders* and tapping to their creativity. Herrmann [29] defines a methodological approach, called the Socio-Technical Walkthrough, to take the multitude of aspects around customer needs into consideration and to make them the subject of communication and negotiation. Unfortunately, such informal methods do not scale well, are costly, and often require expert guidance. To such limitations of workshop-style requirements elicitation, Horkoff and Maiden [31] introduce methods to stimulate creativity and capture creative output in a structured form which might be better amenable to automated analysis and later development. Burnay et al. [9] define creativity triggers for requirements elicitation. Creativity triggers aid stakeholders and engineers to discover new requirements associated with qualities and traits of a product, such as convenience and completeness. In a recent work, Fariha et al. [23] propose a stakeholder collaboration platform with a proactive recommendation model to further automate the elicitation of requirements and reduce the economic, time, and geographical pressures that are required for traditional requirements elicitation workshops [31].

Our approach focuses on better involving *engineers* in the requirements shaping process, and giving them the freedom to be creative by promoting opinions and intuitions developed through exploration to first-class citizens. Nevertheless, the above methods could complement to our work, potentially involving client-side technical experts into the requirements formulation process.

Requirements evolution. A substantial body of knowledge has been dedicated to supporting the incorporation of emerging new knowledge in requirements by their evolution [21]. For example, Grubb and Chechik [27] link stakeholder intentions with technical requirements through goal models [3], and use what-if simulations to determine the impact of requirement change. Approaches that systematically evaluate the impact of requirements evolution are useful complements to our approach, e.g., to explore the least intrusive evolution path that aligns with the alternative design recommended by the engineers. Appropriate semantics have been developed in [28].

Formal codification of requirements in an effort to render them more amenable to automated reasoning has been a research area of particular interest. For example, DeVries and Cheng [18] detect incomplete requirements decompositions using symbolic analysis of hierarchical requirements models at design time. Chechik and Gannon [10] provide a framework for lightweight assessment of consistency between requirements and detailed design using the SCR specification language. To deal with informal requirements, Li et al. [40] transform them to formal ones, and subsequently, to software specifications. An ontology of requirements and a formal requirements modeling language support the representation of functional and extra-functional requirements, which are later refined using a set of semantically sound refinement operators.

Our approach focuses on human faculties and the formalization of subjective opinions, to potentially channel into such methods. By that, our approach eliminates the need for some inconvenient assumptions of formalization-focused requirements methods, e.g., the feasibility of expressing requirements by formal notation, which is certainly challenged by the presence of non-technical experts.

8 CONCLUSION

In this paper, we presented an approach to drive requirements evolution by expert knowledge that might deviate from requirements. Our approach relies on the quantified confidence of engineers in their own design, thereby indicating to the requirements engineer when it is likely that requirements must be re-assessed. We chose subjective logic as the sound formal underpinning of our approach and separated its key parameters into contextual, subject, and objective MDE concerns, such as model consistency and correctness. Leveraging some well-known theorems from model-driven engineering, we make a link between engineers' beliefs and model consistency; and additional links with model correctness and the certainty of engineers' opinions.

Our approach fosters more agile requirements evolution practices by tapping into engineering knowledge, shortening the loop of noticing the need for evolution, and providing requirements engineers with tangible alternative designs to propose to stakeholders.

REFERENCES

[1] Ernest Wilcox Adams. 1996. *A Primer of Probability Logic.* Center for the Study of Language and Inf.

[2] Lars Almefelt, Fredrik Berglund, Patrik Nilsson, and Johan Malmqvist. 2006. Requirements management in practice: findings from an empirical study in the automotive industry. *Research in Engineering Design* 17, 3 (2006), 113–134.

[3] Daniel Amyot, Sepideh Ghanavati, Jennifer Horkoff, Gunter Mussbacher, Liam Peyton, and Eric Yu. 2010. Evaluating goal models within the goal-oriented requirement language. *Int J Intell Syst* 25, 8 (2010), 841–877.

[4] Muhammad Waseem Anwar, Federico Ciccozzi, and Alessio Bucaioni. 2023. Enabling Blended Modelling of Timing and Variability in EAST-ADL. In *Proc. of the ACM Intl. Conf. on Software Language Engineering (SLE 2023)*. ACM, 169–180.

[5] B Barragáns-Martínez, JJ Pazos-Arias, and A Fernández-Vilas. 2004. On measuring levels of inconsistency in multi-perspective requirements specifications. In *Proc. of the 1st Conf. on the Principles of Software Engineering (PRISE'04)*. 21–30.

[6] Bruno Barroca, Thomas Kühne, and Hans Vangheluwe. 2014. Integrating Language and Ontology Engineering. In *Proc. of the 8th Workshop on Multi-Paradigm Modeling (CEUR Workshop Proceedings, Vol. 1237)*. CEUR-WS.org, 77–86.

[7] B.W. Boehm. 1991. Software risk management: principles and practices. *IEEE Software* 8, 1 (1991), 32–41. https://doi.org/10.1109/52.62930

[8] Lola Burgueño, Paula Muñoz, Robert Clarisó, Jordi Cabot, Sébastien Gérard, and Antonio Vallecillo. 2023. Dealing with Belief Uncertainty in Domain Models. *ACM Trans. Softw. Eng. Methodol.* 32, 2 (2023). https://doi.org/10.1145/3542947

[9] Corentin Burnay, Jennifer Horkoff, and Neil Maiden. 2016. Stimulating Stakeholders' Imagination: New Creativity Triggers for Eliciting Novel Requirements. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 36–45.

[10] M. Chechik and J. Gannon. 2001. Automatic analysis of consistency between requirements and designs. *IEEE Trans. on Soft. Eng.* 27, 7 (2001), 651–672.

[11] Lukasz Cyra and Janusz Górski. 2008. Expert assessment of arguments: A method and its experimental evaluation. In *Computer Safety, Reliability, and Security: 27th International Conference, SAFECOMP 2008*. Springer, 291–304.

[12] Kyanna Dagenais. 2024. Towards Model Repair by Human Opinion-Guided Reinforcement Learning. In *Proceedings of the 27th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. ACM.

[13] Kyanna Dagenais and Istvan David. 2024. Opinion-Guided Reinforcement Learning. arXiv:2405.17287

[14] Istvan David et al. 2016. Towards Inconsistency Tolerance by Quantification of Semantic Inconsistencies. In *Proceedings of the 1st International Workshop on Collaborative Modelling in MDE*, Vol. 1717. CEUR-WS.org, 35–44.

[15] Istvan David, Joachim Denil, Klaas Gadeyne, and Hans Vangheluwe. 2016. Engineering Process Transformation to Manage (In)consistency. In *Proc. of the 1st Intl. Workshop on Collaborative Modelling in MDE*, Vol. 1717. CEUR-WS.org, 7–16.

[16] Istvan David, Bart Meyers, Ken Vanherpen, Yentl Van Tendeloo, Kristof Berx, and Hans Vangheluwe. 2017. Modeling and Enactment Support for Early Detection of Inconsistencies in Engineering Processes. In *Proceedings of MODELS 2017 Satellite Event (CEUR Workshop Proceedings, Vol. 2019)*. CEUR-WS.org, 145–154.

[17] Istvan David, Hans Vangheluwe, and Eugene Syriani. 2023. Model consistency as a heuristic for eventual correctness. *Journal of Comp. Lang.* 76 (2023), 101223. https://doi.org/10.1016/j.cola.2023.101223

[18] Byron DeVries and Betty H. C. Cheng. 2016. Automatic detection of incomplete requirements via symbolic analysis. In *Proc. of the ACM/IEEE 19th Intl. Conf. on Model Driven Engineering Languages and Systems (MODELS'16)*. ACM, 385–395.

[19] Lian Duan, Sanjai Rayadurgam, Mats Heimdahl, Oleg Sokolsky, and Insup Lee. 2016. Representation of Confidence in Assurance Cases Using the Beta Distribution. In *IEEE 17th Intl. Symp. on High Assurance Systems Engineering*. 86–93.

[20] Christof Ebert and Jozef De Man. 2005. requirements uncertainty: influencing factors and concrete improvements. In *Proceedings of the 27th International Conference on Software Engineering*. ACM, 553–560.

[21] Neil Ernst, Alexander Borgida, Ivan J. Jureta, and John Mylopoulos. 2014. *An Overview of Requirements Evolution*. Springer, 3–32.

[22] Kweku Ewusi-Mensah. 2003. *Software development failures*. MIT Press.

[23] Asma Fariha, Sanaa Alwidian, and Akramul Azim. 2023. Towards Requirements Specification Collaboration Forum for Embedded Software Systems. In *ACM/IEEE Intl. Conf. on Model Driven Eng. Lang. and Sys. Companion (MODELS-C)*. 312–317.

[24] D. Méndez Fernández et al. 2017. Naming the pain in requirements engineering. *Empirical Software Engineering* 22, 5 (2017), 2298–2338.

[25] P. Gardenfors and N.-E. Sahlin. 2005. *Unreliable Probabilities, Risk Taking, and Decision Making*. Springer, 11–29.

[26] Fabian Gilson and Vincent Englebert. 2011. Rationale, decisions and alternatives traceability for architecture design. In *Proceedings of the 5th European Conference on Software Architecture: Companion Volume (ECSA '11)*. ACM.

[27] Alicia M. Grubb and Marsha Chechik. 2016. Looking into the Crystal Ball: Requirements Evolution over Time. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 86–95. https://doi.org/10.1109/RE.2016.45

[28] Alicia M. Grubb and Marsha Chechik. 2018. BloomingLeaf: A Formal Tool for Requirements Evolution Over Time. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*. 490–491. https://doi.org/10.1109/RE.2018.00067

[29] Thomas Herrmann. 2009. *Systems Design with the Socio-Technical Walkthrough*. IGI Global, 336–351. https://doi.org/10.4018/978-1-60566-264-0.ch023

[30] M. Hoffmann, N. Kuhn, M. Weber, and M. Bittner. 2004. Requirements for requirements management tools. In *Intl. Requirements Eng. Conf., 2004*. 301–308.

[31] J. Horkoff and N. Maiden. 2015. Creativity and conceptual modeling for requirements engineering. *CEUR Workshop Proceedings* 1342 (2015), 62–68.

[32] Robbert Jongeling and Antonio Vallecillo. 2023. Uncertainty-aware consistency checking in industrial settings. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 73–83.

[33] Audun Jøsang. 2016. *Subjective Logic*. Springer.

[34] Audun Jøsang and Viggo A. Bondi. 2000. Legal reasoning with subjective logic. *AI and Law* 8, 4 (2000), 289–315.

[35] Audun Jøsang, Paulo C.G. Costa, and Erik Blasch. 2013. Determining model correctness for situations of belief fusion. In *Proceedings of the 16th International Conference on Information Fusion*. 1886–1893.

[36] Ankur Joshi et al. 2015. Likert scale: Explored and explained. *British Journal of Applied Science & Technology* 7, 4 (2015), 396.

[37] Shraddha Joshi and Joshua D. Summers. 2015. Requirements Evolution: Understanding the Type of Changes in the Requirement Document of Novice Designers. In *ICoRD'15 – Research into Design Across Boundaries Volume 2*. Springer, 471–481.

[38] Leslie Lamport. 1977. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering* SE-3, 2 (1977), 125–143.

[39] C. Lange, M. R. V. Chaudron, J. Muskens, L. J. Somers, and H. M. Dortmans. 2003. An empirical investigation in quantifying inconsistency and incompleteness of UML designs. In *Incompleteness of UML Designs, Proc. Workshop on Consistency Problems in UML-based Software Development*.

[40] Feng-Lin Li et al. 2015. From Stakeholder Requirements to Formal Specifications Through Refinement. In *Requirements Engineering: Foundation for Software Quality*. Springer, 164–180.

[41] Yue Ma, Guilin Qi, Pascal Hitzler, and Zuoquan Lin. 2007. Measuring Inconsistency for Description Logics Based on Paraconsistent Semantics. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. Springer, 30–41.

[42] Neil Maiden, James Lockerbie, Debbie Randall, Sean Jones, and David Bush. 2007. Using Satisfaction Arguments to Enhance i*Modelling of an Air Traffic Management System. In *15th IEEE Intl. Requirements Engineering Conf.* 49–52.

[43] Francesco Margoni and Neil Walkinshaw. 2023. Subjective Logic as a Complementary Tool to Meta-Analysis to Transparently Address Second-Order Uncertainty in Research Findings. (2023).

[44] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *ACM Comput. Surv.* 37, 4 (2005), 316–344.

[45] Rakshit Mittal, Raheleh Eslampanah, Lucas Lima, Hans Vangheluwe, and Dominique Blouin. 2023. Towards an Ontological Framework for Validity Frames. In *ACM/IEEE Intl. Conf. on Model Driven Eng. Lang. and Sys. Companion*. 801–805.

[46] Francisco J Navarrete and Antonio Vallecillo. 2021. Introducing subjective knowledge graphs. In *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 61–70.

[47] Ikujirō Nonaka and Hirotaka Takeuchi. 2007. The knowledge-creating company. *Harvard Business Review* 85, 7/8 (2007), 162.

[48] Klaus Pohl, Günter Böckle, and Frank van der Linden. 2005. *Software Product Line Engineering*. Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-28901-1

[49] Paul Ralph. 2013. The illusion of requirements in software development. *Requirements Engineering* 18, 3 (2013), 293–296.

[50] Marc HJ Romanycia and Francis Jeffry Pelletier. 1985. What is a heuristic? *Computational Intelligence* 1, 1 (1985), 47–58.

[51] Rick Salay et al. 2013. Managing requirements uncertainty with partial models. *Requirements Engineering* 18, 2 (2013), 107–128.

[52] Douglas C Schmidt. 2006. Model-driven engineering. *Computer* 39, 2 (2006), 25.

[53] S Sedhumadhavan and E Niranjana. 2017. An Analysis of Path Planning for Autonomous Motorized Robots. *International Journal of Advance Research, Ideas and Innovations in Technology* 3, 6 (2017), 1234–1257.

[54] Javier Troya, Nathalie Moreno, Manuel F. Bertoa, and Antonio Vallecillo. 2021. Uncertainty representation in software models: a survey. *Soft. Sys. Mod.* 20, 4 (2021), 1183–1213.

[55] Wade M Vagias. 2006. Likert-type scale response anchors. *Clemson International Institute for Tourism & Research Development, Department of Parks, Recreation and Tourism Management. Clemson University* (2006).

[56] Ragnhild Van Der Straeten, Tom Mens, Jocelyn Simmonds, and Viviane Jonckers. 2003. Using Description Logic to Maintain Consistency between UML Models. In *«UML» 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference (LNCS, Vol. 2863)*. Springer, 326–340.

[57] Hans Van Ditmarsch, Wiebe van der Hoek, Joseph Y Halpern, and Barteld Kooi. 2015. *Handbook of epistemic logic*. College Publications.

[58] Ken Vanherpen, Joachim Denil, Istvan David, Paul De Meulenaere, Pieter J. Mosterman, Martin Torngren, Ahsan Qamar, and Hans Vangheluwe. 2016. Ontological reasoning for consistency in the design of cyber-physical systems. In *2016 1st International Workshop on Cyber-Physical Production Systems*. IEEE, 1–8. https://doi.org/10.1109/CPPS.2016.7483922

[59] Neil Walkinshaw and Martin Shepperd. 2020. Reasoning about Uncertainty in Empirical Results. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering (EASE '20)*. ACM, 140–149.

[60] Matthias Weber and Joachim Weisbrod. 2002. Requirements engineering in automotive development-experiences and challenges. In *Proc. IEEE Joint Intl. Conf. on Requirements Engineering*. 331–340.