Empirical Software Engineering (2019) 24:2509–2539
https://doi.org/10.1007/s10664-019-09693-x

An empirical study on the potential usefulness of domain models for completeness checking of requirements

Chetan Arora[1,2] · **Mehrdad Sabetzadeh**[1] · **Lionel C. Briand**[1]

Published online: 18 April 2019

Abstract

Domain modeling is a common strategy for mitigating incompleteness in requirements. While the benefits of domain models for checking the completeness of requirements are anecdotally known, these benefits have never been evaluated systematically. We empirically examine the potential usefulness of domain models for detecting incompleteness in natural-language requirements. We focus on requirements written as "shall"-style statements and domain models captured using UML class diagrams. Through a randomized simulation process, we analyze the sensitivity of domain models to omissions in requirements. Sensitivity is a measure of whether a domain model contains information that can lead to the discovery of requirements omissions. Our empirical research method is case study research in an industrial setting. We have experts construct domain models in three distinct industry domains. We then report on how sensitive the resulting models are to simulated omissions in requirements. We observe that domain models exhibit near-linear sensitivity to both unspecified (i.e., missing) and under-specified requirements (i.e., requirements whose details are incomplete). The level of sensitivity is more than four times higher for unspecified requirements than under-specified ones. These results provide empirical evidence that domain models provide useful cues for checking the completeness of natural-language requirements. Further studies remain necessary to ascertain whether analysts are able to effectively exploit these cues for incompleteness detection.

Keywords Requirements quality assurance · Requirements completeness · Natural-language requirements · Domain modeling · Case study research

1 Introduction

Checking the completeness of requirements is one of the most important and yet one of the most challenging aspects of requirements validation (Basili and Weiss 1981; Davis 1990;

Communicated by: Daniel Berry

✉ Chetan Arora
arora@svv.lu

Extended author information available on the last page of the article.

Schneider et al. 1992; Pohl 1993; Zowghi and Gervasi 2003b). Requirements completeness is often characterized along two dimensions: *internal* and *external* (Zowghi and Gervasi 2003a). Internal completeness is concerned with ensuring that a given requirements specification is closed with respect to the statements and inferences that can be made solely based on that specification (Jaffe et al. 1991). Internal completeness criteria include, among others, absence of to-be-defined (TBD) items (Boehm 1984). External completeness is concerned with ensuring that all the information that is pertinent to the definition of a system is found within the specification (Zowghi and Gervasi 2003a). External completeness is a relative notion and has to be determined vis-à-vis external sources of information. These sources may be either humans (stakeholders) (Davis et al. 1993; Moody et al. 1998) or development artifacts such as higher-level requirements (Costello and Liu 1995), models (Gigante et al. 2015; Geierhos and Bäumer 2016), and existing system documentation (Ferrari et al. 2014). External completeness criteria include, among others, absence of missing system functions (Boehm 1984).

Our work in this article focuses on *external completeness*. A fundamental property of external completeness is that it cannot be ascertained in absolute terms. This is because one can never be entirely sure that all the relevant external sources have been identified, or that the identified external sources themselves are indeed complete. Despite this limitation, one can implement measures for *improving* (but not guaranteeing) the external completeness of requirements. One such measure is *domain modeling* (Zowghi and Gervasi 2003a; Ferrari et al. 2014). A domain model is an explicit representation of the salient concepts in an application domain and the relations between these concepts (Evans 2004). Depending on the context, one may choose among several alternative notations for domain modeling. These notations include ontology languages such as OWL (W3C OWL Working Group 2012), entity-relationship (ER) models, and UML class diagrams (Ambler 2004; Holt et al. 2011).

Domain models, by virtue of being structured and succinct, help analysts capture the domain concepts and relations in a reasonably complete manner. Further, and by virtue of being predominantly graphical, domain models are accessible to domain experts, provided that the experts are sufficiently trained in the modeling notation being used. This means that the experts can more easily review a domain model for completeness than lengthy requirements documents.

One would expect that the concepts and relations in a domain model should be referred to in the requirements at least once, but potentially multiple times. A domain model element that is not referred to in the requirements may be an indication of incompleteness and thus warrants further investigation. Although this way of using a domain model for completeness checking of requirements is intuitive and derived from common sense (Zowghi and Gervasi 2003a; Kaiya and Saeki 2005), there is little empirical research that examines in a systematic manner how useful domain models can be for detecting incompleteness in requirements.

In this article, we present an empirical study aimed at investigating the potential usefulness of domain models for detecting incompleteness in *natural-language (NL) requirements*. Our interest in NL requirements is motivated by their prevalent use in industry (Mich et al. 2004; Pohl and Rupp 2011). Checking the completeness of NL requirements is particularly challenging, noting that NL requirements specifications may constitute hundreds or thousands of statements. This makes it important to develop a more detailed understanding of mechanisms through which one can identify incompleteness issues in NL requirements. One interesting mechanism to study for this purpose is domain modeling. This is due to both the widespread use of domain models, and the fact that domain model construction often

follows established guidelines (Larman 2004; Pohl and Rupp 2011; Whittle et al. 2014). The latter factor leads to a reasonable degree of consistency in how analysts, irrespective of their application domain, go about building domain models. Consequently, empirical observations about the usefulness of domain models, including for completeness checking of NL requirements, are likely to generalize beyond the immediate context where the observations are made.

Both NL requirements and domain models are broad terms, each with multiple realizations. NL requirements may come in a variety of forms, e.g., "shall"-style statements, use cases, user stories and feature lists (Pohl and Rupp 2011). Similarly, and as already mentioned, domain models may be expressed in different notations with different capabilities and characteristics. Our work in this article necessitates that we pick specific interpretations for NL requirements and domain models, since different interpretations may lead to different empirical outcomes.

For this article, we take NL requirements to mean *"shall"-style statements* (for short, *shall requirements*). Shall requirements are common practice in industry (Whittle et al. 2009). For domain modeling, we employ *UML class diagrams*. Class diagrams are a popular notation for building conceptual representations of a domain (Reggio et al. 2014). We concern ourselves exclusively with *functional requirements*, noting that domain models have a functional nature (Lindland et al. 1994), and further, all the requirements specifications in our study are functional ones (see Section 3). The general research question we tackle in this article can thus more precisely be stated as follows:

> *RQ: Are domain models represented as UML class diagrams potentially useful for checking the completeness of functional requirements expressed as shall statements?*

Example We illustrate through an example the relationship between shall requirements and domain models represented as class diagrams. Using this example, we explain in a concrete manner how a domain model can assist with checking the completeness of requirements.

Figure 1a shows a subset of the requirements for a simulator tool in the aerospace domain. The requirements document from which this subset was taken is the subject of one

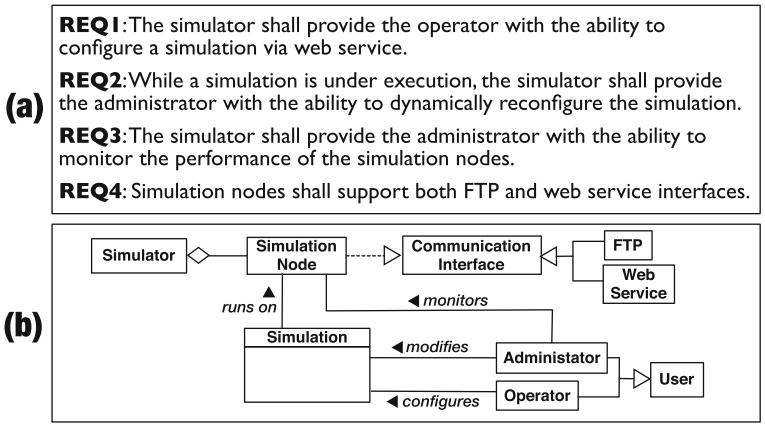

Fig. 1 **a** Examples of shall requirements; **b** Example domain model represented as a UML class diagram

of our case studies in Section 3. The requirements in Fig. 1a have been slightly altered from their original form to facilitate discussion and preserve confidentiality. Figure 1b shows the domain model fragment pertinent to the requirements of Fig. 1a. This fragment comes from a domain model built by subject-matter experts. Some small modifications were made to this model fragment to align it with the altered requirements. For the purposes of our illustration, we assume that both the requirements and the domain model in Fig. 1 are *complete*, i.e., free from any incompleteness issues.

Several elements of the domain model of Fig. 1b have correspondences in the requirements of Fig. 1a. For example, the concepts of *Simulator* and *Simulation* are shared between both representations. The correspondences are not always at a lexical level. For example, consider REQ2. At the level of requirements, the conceptual link between *Administrator* and *Simulation* is for performing a specific action (dynamic reconfiguration). In the domain model, this link is covered by an abstract association, labeled *modifies*, which does not directly refer to the above action.

Domain models and requirements have different expressive powers, and not all the information that can be inferred from domain models and requirements is overlapping or equivalent (Larkin and Simon 1987). There may be information that is present exclusively in the domain model – let us denote this information by D_{only} – as well as information that is present exclusively in the requirements – let us denote this by R_{only}. Assuming that the requirements are complete, as we do in our example, one may expect that D_{only} should be empty. In practice nevertheless, the domain model may contain information that analysts, by choice, would leave *implicit (tacit)* in shall requirements. For example, the abstract concepts of *User* and *Communication Interface* do not appear in the requirements of Fig. 1a. Similarly, absent from the requirements is the domain model association between *Simulator* and *Simulation Node*. While the information in D_{only} is important for correctly interpreting the requirements, the engineers often opt for more suitable means than shall statements, e.g., a glossary, for expressing such information. Shall requirements should thus not automatically be treated as incomplete merely because one can find additional information in the domain model.

As for R_{only}, the information may have been left out of the domain model for various reasons, e.g., lack of expressiveness in the domain modeling notation (class diagrams in our case), or the information being too obvious, too detailed or too abstract for the domain model. We note again that we are arguing under the assumption that the domain model is complete. For example, the model of Fig. 1 does not convey the constraint stated in REQ1 that the simulations shall be configured "via a web service", although the concepts of *Simulation* and *Web Service* are both present in the model; the subject-matter expert found this constraint too detailed for the domain model. Similarly, the condition that the functionality in REQ2 should be rendered "while a simulation is under execution" is not captured in the model; this condition concerns system behavior and is not readily expressible in (simple) class diagrams.

We now explain how an analyst can use a reasonably complete domain model for detecting incompleteness in requirements. Suppose that *FTP* has been unintentionally omitted from REQ4 of Fig. 1a, i.e., REQ4 reads as: "Simulation nodes shall support the web service interface." An inspection of the domain model in Fig. 1b should raise suspicion of incompleteness, since *FTP* appears in the model but not in the requirements. The domain model is not always as sensitive to omissions. For example, had "web service" been left out from REQ4 instead, a simple cross-checking of the domain model against the requirements would have been unlikely to hint at incompleteness: the term "web service" appears in REQ1 and thus remains present even after it is removed from REQ4. In this case, it would take more

omissions, i.e., a larger degree of incompleteness, before the domain model can alert the analyst to a potential incompleteness. Specifically, it would take REQ1 or at least the "via web service" constraint of REQ1 to be missing, before the domain model would sense the omission.

The associations in a domain model may too be used for identifying incompleteness issues. For example, suppose that REQ2 is missing from the requirements. Although *Simulation* and *Administrator* both appear in other requirements, the fact that these concepts are associated in the domain model (the *modifies* association) but never co-located in the same requirement would be cause for suspecting incompleteness. Attributes can be used in a similar manner for completeness checking, although not illustrated here for succinctness.

As one can see from our illustration of Fig. 1, domain concepts may be referred to several times in the requirements. For example, *Simulator* is referred to three times in the requirements (REQ1–3). The same applies to other domain model element types (relations and attributes), although not illustrated here. The number of times a domain model element is referred to in the requirements has no bearing on the domain model itself. However and as exemplified above, the higher the level of repetition, the less sensitive the domain model becomes to omissions. Repeated references thus need to be considered as an influencing factor when examining the usefulness of domain models for checking the completeness of requirements.

Contribution In our illustrating example, we motivated two main factors affecting the usefulness of a domain model for checking the completeness of NL requirements. These factors are: (1) the level of content overlap between the requirements and the domain model, and (2) how frequently the elements of the domain model are referred to in the requirements. How these factors come together to determine the sensitivity of domain models to omissions in NL requirements is a topic that needs to be investigated empirically in realistic contexts. To this end, we conducted three case studies over industrial requirements in different application domains. In all cases, the requirements were written as shall statements. We had subject-matter experts build a domain model in each case study. As part of this process, we established fine-grained traceability between the domain model elements and the requirements.

Using the resulting domain models and their traceability to the requirements, we then analyzed the sensitivity of the domain models to omissions in the requirements. Stated otherwise, we analyzed whether one has a chance of detecting omissions via inspecting what elements in the domain model are missing in the requirements. The strategy employed for this purpose is a *randomized simulation process*, through which we randomly removed entire requirements or parts thereof from the original set of requirements. Subsequently, using the traceability information between the requirements and the domain model, we identified domain model elements that were no longer supported (covered) in the requirements after the omissions had been applied. For a domain model to have a chance of being useful for identifying incompleteness in the requirements, the number of unsupported domain model elements should increase rapidly as larger omissions are seeded into the requirements.

Our case studies yield consistent conclusions. Most importantly, we observe that domain models exhibit near-linear sensitivity to both unspecified requirements, i.e., requirements that are missing, and under-specified requirements, i.e., requirements whose details, e.g., constraints and conditions have not been provided. An example constraint in Fig. 1a is "via web service" (REQ1), and an example condition is "While a simulation is under execution" (REQ2). The level of sensitivity that domain models show to omissions is more than four

times higher when whole requirements are removed, compared to when requirements details are removed.

Despite being promising, our results do not automatically lead to the conclusion that domain models are useful instruments for completeness checking of requirements by analysts. What our current study shows is that domain models are sufficiently sensitive to omissions in NL requirements. We nevertheless do not look into whether analysts can successfully detect the omissions for which the domain model holds the right information cues, nor whether the extra information conveyed by the domain model is actually essential for completing the requirements. Further, while done in an industrial setting, our case studies had to be scoped to ensure that the subject-matter experts involved would be able to perform their tasks without time pressure. In light of the above, future user studies remain necessary for measuring how well practitioners can exploit the potential offered by domain models for completeness checking of requirements.

Structure Section 2 outlines background and compares with related work. Section 3 describes the design of our empirical study. Section 4 reports the results of the study. Section 5 discusses threats to validity. Section 6 concludes the article.

2 Background and Related Work

In this section, we outline related research strands concerned with requirements (in)completeness. We organize our discussion under three headings, covering the foundations (Section 2.1), approaches for completeness checking of NL requirements (Section 2.2), and empirical studies on requirements completeness (Section 2.3).

2.1 Foundations

2.1.1 General Definitions

In his pioneering work, Boehm (1984) defines a requirements specification to be complete if there are no (1) to-be-determined items, (2) nonexistent references such as inputs and outputs, (3) missing specification items such as interface specifications, (4) missing functions, and (5) missing product information. The IEEE 29148:2011 (IEEE 2011) standard proposes a more overarching definition: A requirements statement is complete if it "needs no further amplification because it is measurable and sufficiently describes the capability and characteristics to meet the stakeholder's need". A requirements specification (set of requirements) is complete if the set "needs no further amplification because it contains everything pertinent to the definition of the system or system element being specified. In addition, the set contains no to-be-defined, to-be-specified, or to-be-resolved clauses".

Zowghi and Gervasi (2003b) distinguish the notions of *internal* and *external* completeness for requirements. We already discussed this distinction in Section 1. Ferrari et al. (2014) elaborate external completeness into *backward* and *forward*. Backward completeness is measured against the body of knowledge established *prior to* the development of a requirements specification. An example of such knowledge represented as explicit artifacts would be transcripts from stakeholder interviews. Forward completeness is, in contrast, measured against the (accumulated) body of knowledge *after* a requirements specification has been produced. Examples of such knowledge would be late-stage requirements models and design models.

A question that arises in our work is whether measuring the completeness of NL requirements against a domain model constitutes backward or forward checking. Since domain modeling can potentially span the entire requirements engineering process (Larman 2004), both backward and forward checking are possibilities. In our case studies of Section 3, the domain models are at the level of maturity that one would expect at late-stage requirements engineering. Our mode of using domain models thus coincides with *forward checking* of NL requirements. In general, forward checking is more plausible than backward checking for NL requirements. This is because, usually, NL requirements specifications are the contractual basis for a system to be built, and thus the first artifacts to be produced. Most development activities, including domain modeling, are often deferred to *after* the requirements have met the contractual necessities; there is little (financial) justification in building a domain model when the system to be built has not been signed off on. This makes backward checking of NL requirements inapplicable in many practical situations.

2.1.2 Completeness of Conceptual Models

There is considerable research on the completeness of conceptual models, including conceptual models of requirements. While our work in this article is motivated by improving the completeness of NL requirements rather than that of models of requirements, the instrument whose usefulness we are investigating for this purpose is a (domain) model. As we alluded to in Section 1 and will explain further in Section 3, our case studies involve having subject-matter experts construct domain models that are as complete as possible. Using existing work on the completeness of conceptual models, we explain below what a "complete domain model" means in our context.

Lindland et al. (1994) define a model to be *(semantically) complete* with respect to a given domain if the model contains all the correct and relevant statements in that domain. Lindland et al. (1994) further introduce the notion of *feasible completeness*, where a conceptual model contains only a subset of the statements in a domain. A conceptual model is feasibly complete with respect to a domain, if any further enhancement of the model is deemed less beneficial than accepting the model as-is.

The notion of feasible completeness is important for domain models: As illustrated over the example of Fig. 1, experts make conscious choices as to what information to include in and exclude from the domain model. In a real setting, one can thus expect an ideal domain model to be only feasibly complete. In our case studies of Section 3, feasible completeness is what we aim to achieve for the domain models built.

2.2 Checking the Completeness of NL Requirements

One can employ various strategies for checking the completeness of requirements. These include (1) attempting to mold the requirements into structured templates (Pohl and Rupp 2011; Eckhardt et al. 2016) or models (Heimdahl and Leveson 1996; Heitmeyer et al. 610; Schuette and Rotthowe 1998); (2) engaging closely with the stakeholders during requirements elicitation and validation (Moody et al. 1998; Salger et al. 2009); (3) following a multi-perspective elicitation approach (Nuseibeh et al. 1994) and later reconciling the perspectives (Easterbrook et al. 2005; Sabetzadeh and Easterbrook 2006; Khatwani et al. 2017; Dalpiaz et al. 2018); (4) employing formal synthesis and verification (Alrajeh et al. 2012; Zhou et al. 2014); (5) cross-validating requirements against other development artifacts, e.g., higher-level requirements (Costello and Liu 1995), system documents (Ferrari et al. 2014) and ontologies (Kaiya and Saeki 2005; Gigante et al. 2015; Geierhos and Bäumer

2016); and (6) requirements reviews and inspections (Basili et al. 1996; Fusaro et al. 1997; Porter and Votta 1998; Thelin et al. 2003).

Below, we review existing work on completeness checking of *NL requirements*, focusing on external completeness.

Costello and Liu (1995) propose to assess the completeness of requirements by checking whether all high-level requirements have been decomposed into lower-level ones. (Ferrari et al. 2014) develop an approach based on natural language processing (NLP) for completeness checking of requirements. The approach works by comparing the terminology used in a requirements document against those used in legacy system specifications and stakeholder interviews. Dalpiaz et al. (2018) employ a combination of NLP and visualization for extracting and contrasting concepts that originate from different stakeholder viewpoints. The contrasts are treated as potential incompleteness issues. Kaiya and Saeki (2005) propose to check the completeness of requirements by (manually) mapping requirements concepts to the concepts in a domain ontology. The unmapped concepts in the ontology are taken as indicators of potential incompleteness. Using NLP, Gigante et al. (2015) extract from a set of requirements and an external source – in their case, higher-level requirements – systematic information in the form of (subject, predicate, object) triplets. They then compare the triplets from the two sources to determine how complete the given set of requirements is. Kamalrudin et al. (2011) employ NLP to extract from textual requirements lightweight models known as essential use case models (Constantine and Lockwood 1999). The completeness of these models is then analyzed by comparing them against a set of best practices specified in a pattern library.

The above approaches work by either directly or indirectly matching (cross-validating) the constituents of requirements sentences, e.g., noun phrases and verb phrases, against an external source. Our evaluation of the usefulness of a domain model as an external source for completeness checking draws on the same general principle. With the exception of Kaiya and Saeki (2005), however, none of the above approaches measure requirements completeness against a domain model. As for the work of Kaiya and Saeki's, there is no empirical evaluation provided on completeness checking. Further and more importantly, the focus of our work differs from that of Kaiya and Saeki's: they propose a simple metric for requirements completeness, taking for granted that there is a relationship between NL requirements and domain models. As we argued in Section 1, however, the exact nature of this relationship has never been investigated. Our work sheds empirical insights on the characteristics of the relationship between NL requirements and domain models, and examines the usefulness of this relationship for detecting omissions in requirements.

2.3 Empirical Studies on Requirements Completeness

There are controlled experiments that compare the level of requirements completeness achieved by different requirements specification approaches. Yadav et al. (1988) compare the completeness of requirements specified using data flow diagrams (Gane and Sarson 1979) and the integrated definition method (Bravoco and Yadav 1985). No statistically-significant results are obtained for completeness. España et al. (2009) compare use case models (Cockburn 1997) against an information systems development approach, called communication analysis (España et al. 2009). They conclude that communication analysis yields more complete requirements specifications with respect to a reference model. The notion used for measuring completeness is feasible functional completeness – a restriction of Lindland et al.'s definition of feasible completeness (see Section 2.1.2) to functional requirements. Menzel et al. (2010) propose a functional requirements specification approach

based on structured templates for interfaces, inputs, outputs and system function behaviors. They compare this functional approach against a use-case-based approach (Koenig et al. 2007) using a notion of completeness defined around goals and their associated information elements. They observe that both approaches have their own merits depending on the type of information being specified.

Our work differs from the above in three main ways: First, our study context is different, centering around shall requirements and domain models expressed as UML class diagrams. Second, our goal is not to compare alternative specification approaches. Instead, we examine how a complementary approach, namely domain modeling, contributes to improving requirements completeness. Third, and from the perspective of empirical foundations, our work builds on case study research in an industrial setting rather than controlled experiments.

3 Study Design

In this section, we describe the design of our case studies. There are three case studies in total, which we refer to as Case A, Case B, and Case C. Case A concerns a simulator module for aerospace applications. The illustrative requirements in Fig. 1 come from Case A. Case B concerns a sensor platform for cyber-physical systems, and Case C, a content management system for safety assurance purposes. The functional requirements in all three case studies are expressed as shall statements. Throughout the remainder of the article, we take the term "requirement" to mean an individual shall statement. Case A has 163 functional requirements, Case B has 212, and Case C has 110. Our analysis uses only a subset of these requirements, as we explain momentarily.

The main components of our study are: (1) constructing feasibly complete domain models, (2) establishing traceability between the domain models and the requirements, and (3) simulating potential omissions in the requirements and examining, using the traceability information, whether the domain models contain cues leading to the detection of these omissions.

An important challenge in our study design is that constructing a feasibly complete model for an entire industrial domain requires significant time commitment from subject-matter experts. To ensure that the experts in our study had sufficient time to make the domain models as complete as possible, we had to scope the domain modeling activity. We did so by picking a random subset of the requirements in each case study, and orienting domain model construction around the set of concepts in these requirements. The experts were allowed to add additional concepts to this set as they deemed appropriate. To avoid bias, this scoping was done in a way that the experts could not know, at domain modeling time, which requirements the domain concepts originated from. We describe the scoping process more precisely in Section 3.3.

3.1 Research Questions

RQ1. How sensitive are domain models to omissions in requirements? To study the usefulness of domain models for completeness checking of requirements, we need to determine how closely domain models and requirements relate to one another. RQ1 (answered in Section 4) aims to quantitatively analyze the sensitivity of domain models to omissions in requirements. Sensitivity is a measure of whether a domain model contains the information necessary for inferring the presence of requirements omissions.

RQ2. How is sensitivity related to the intrinsic properties of a requirements document? As we argue in our answer to RQ1, domain models are in general sensitive to omissions in requirements. We nevertheless see a degree of variation in sensitivity across our case studies. RQ2 (also answered in Section 4) was prompted by the variation observed in RQ1. Outside an evaluation setting, one cannot directly analyze sensitivity due to the absence of a gold standard. RQ2 seeks to provide a surrogate mechanism for approximating sensitivity based exclusively on the intrinsic properties of a requirements document and independently of a domain model.

3.2 Case Selection

Our case selection was opportunistic, but subject to certain criteria: First, we were interested in projects whose requirements documents had been already finalized. This criterion was aimed at ensuring that the requirements were as complete as possible. Second, our study requires significant involvement from domain experts for building the domain models and performing other tasks explained later in this section. Reliable access to experts was therefore a critical criterion to fulfill in our study. Finally, we wanted to cover cases from different domains. This is important for improving the external validity of our study, and further to avoid potential bias in our results due to the specificities of a particular domain.

3.3 Data Collection Procedure

Our data collection is aimed at: (1) constructing (feasibly complete) domain models, (2) defining potential but realistic omissions from requirements, (3) tracing requirements to domain model elements, and (4) gathering data that enables us to argue about the representativeness of the results obtained from our case studies. In Case A, two experts were involved in data collection; in each Case B and Case C, there were three experts involved (eight experts across the three case studies). In each case study, at least one of the experts had directly participated in writing the requirements. All the experts had at least five years of domain experience, were familiar with UML, and had built domain models before. Below, we describe the steps of our data collection.

3.3.1 Domain Model Construction

Figure 2 outlines the procedure used for domain model construction. As explained at the beginning of Section 3, we had to scope domain modeling to a subset of the domain concepts. We did so as follows: In each case study, we randomly selected 35 requirements. Let us denote by $\mathcal{R}$ the subset of requirements selected for a given case study. We extracted all the atomic noun phrases (NPs) from the sentences in $\mathcal{R}$. Following object-oriented domain modeling guidelines (Larman 2004), we considered each NP as a candidate domain concept. Without revealing $\mathcal{R}$ to the domain experts, we had them review the NPs extracted from these requirements and select the genuine domain concepts. The experts were allowed to make lexical adjustments to an NP before accepting it as a domain concept. Let us denote by $\mathcal{S}$ the set of domain concepts resulting from the experts reviewing the NPs extracted from $\mathcal{R}$.

To create a domain model, we asked the experts to focus on the concepts in $\mathcal{S}$. This helped ensure that (1) the experts had sufficient time to make the domain model as complete as possible in so far as the concepts in $\mathcal{S}$ are concerned, (2) the concepts in $\mathcal{S}$ are readily traceable to $\mathcal{R}$ (by virtue of how $\mathcal{S}$ was built). The experts were further told that they were

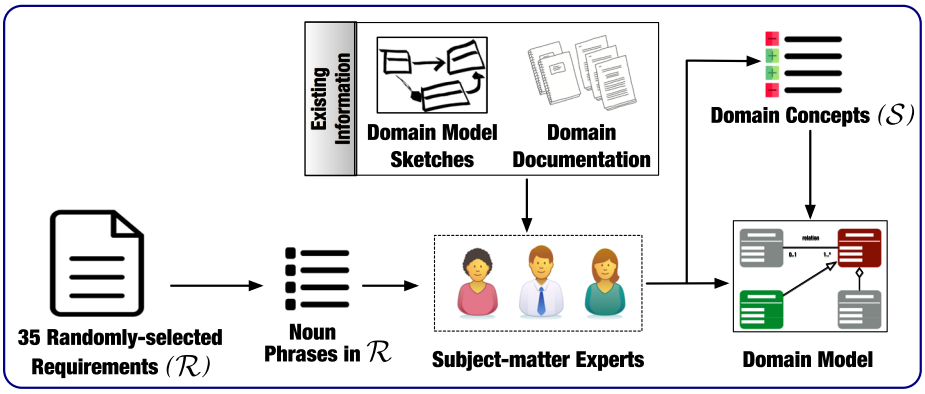

Fig. 2 Procedure for domain model construction. The noun phrases provided to the experts are for scoping domain modeling and ensuring that feasible completeness is achievable

free to introduce any additional concepts they deemed necessary for properly elaborating the concepts in $\mathcal{S}$. Examples of additional concepts introduced by the experts are abstractions such as *User* and *Communication Interface* in the model of Fig. 1b.

The construction of the domain models adhered to established best practices in object-oriented analysis (Larman 2004). The experts followed an exploratory process for domain modeling, relying on both their expertise and the existing development artifacts, e.g., system descriptions and requirements documents (shown as "Domain Documentation" in Fig. 2). Two researchers (the first two authors) acted as facilitators during modeling. Specifically, the researchers helped kick-start the modeling activities by offering the experts a brief refresher training on UML class diagrams and domain modeling. The researchers further answered any questions the experts had about domain modeling choices and best practices as well as the modeling notation. The modeling activities concentrated on specifying the core domain model elements, namely concepts (classes), concept attributes, and relations (associations and generalizations). In our context where the domain models are expressed as class diagrams, one can further provide logical domain constraints using the Object Constraint Language (OCL) (Object Management Group 2004). This was not attempted in our case studies, since our experts were not adequately familiar with OCL. Excluding OCL constraints is a reasonable decision in regard to representativeness, since OCL has not yet been broadly adopted by practitioners (Chimiak-Opoka 2009). We note that the researchers were not physically present during the entire course of modeling activities. In all three case studies, the experts did a portion of the modeling work offline but collaboratively. The results were then provided to the researchers.

As shown in Fig. 2, in all three case studies, partial sketches of a domain model were already existing. For Case A and Case B, the sketches were built on the request of the system clients in order to improve the understandability of certain concepts. In Case B, the sketches were developed mainly for structuring the design process. The experts in our study used these existing sketches as a starting point, pruning what was not related to the concepts in $\mathcal{S}$, and further elaborating the parts pertinent to this subset of concepts.

Domain modeling was concluded before we continued with the remainder of data collection. This was necessary to avoid bias, since the next steps of data collection make the selected subset of requirements, $\mathcal{R}$, known to the experts.

3.3.2 Identifying Omittable Requirement Segments

Answering RQ1 involves simulating realistic requirements omissions. We consider two classes of omissions: (1) omitting requirements in their entirety; and (2) omitting segments of individual requirements. The former represents requirements that are missing from a requirements specification; the latter represents requirements that are present but deficient in their details.

Omitting entire requirements is straightforward as, in none of our case studies, we had requirements that cross-referenced other requirements. There were thus no obvious inter-requirements dependencies. Omitting segments of individual requirements is more involved: Different segments of a requirement may not be removable arbitrarily due to being semantically interdependent. Semantic interdependencies, as we elaborate later in this section, are the logical links between different constituents of a sentence, and are instrumental in making a sentence coherent and meaningful as a whole. Further, even when such interdependencies are accounted for, there is only so much content that can be left out before a requirement becomes trivial and thus easily deemable as incomplete by an expert without any additional instruments such as a domain model.

To simulate the omission of requirement segments in a realistic manner, both semantic interdependencies and the significance of the content of segments need to be considered. We derived the omittable segments as follows: First, two researchers independently reviewed the requirements in each case study, and marked the atomic segments that could potentially be removed. We chose the granularity of an omission to be at least one full noun phrase. The main criterion to decide whether a segment was omittable was the plausibility of the segment being overlooked in realistic conditions. For every segment deemed omittable, each researcher further assigned a semantic type according to the shall-requirement slots defined by the IEEE 29148:2011 standard (IEEE 2011). These slots are: *Subject*, *Object*, *Action*, *Condition*, and *Constraint*. For detailed definitions and examples, consult the standard.[1] As a reliability measure, interrater agreement was computed (see Table 1). An agreement was counted when both researchers found a segment omittable and assigned the same semantic type to it. Other situations counted as disagreements. Differences were reconciled through discussion. The resulting omittable segments were reviewed and approved by the experts in each case study.

Figure 3 illustrates omittable segments in three requirements. For example, by omitting from REQ5 the segments denoted Object_2^5 and Constraint_2^5, we obtain the following: "The simulator shall be able to transfer the simulation execution plan to the user help desk via FTP." Note that not all that can be removed from a grammatical standpoint constitutes a legitimate omittable segment. For example, one can remove "to the user help desk" from REQ5 and still get a meaningful sentence. Nevertheless, forgetting to include the destination of a transfer was found to be unrealistic by the experts. This segment is thus not omittable.

In the next step, the researchers collaborated with the experts to identify the interdependencies between the omittable parts. Our findings about interdependencies are indeed part of our results. Nevertheless, we elect to present these findings here, rather than in Section 4, because our analysis procedure (Section 3.4) requires knowledge of the findings.

[1] Briefly, subjects, actions (verbs), and objects are the core linguistic parts of NL requirements. Conditions are measurable attributes. They further qualify requirements, and may be employed for limiting the options open to a designer, or for expressing information necessary for validation and verification. Constraints restrict the design or implementation of the systems engineering process.

Table 1 Data collection results for the requirements samples

Case	# of Omittable parts		# of interdependencies		Interrater agreement (Cohen's κ)	# of domain model elements (# of tacit elements)		# of trace links
Case A	Requirements	35	Type 1*	13	0.918 (almost perfect agreement)	Concepts	144 (30)	483
	Conditions	5				Attributes	41 (0)	
	Constraints	42	Type 2*	0		Associations	95 (10)	
	Objects	22				Generalizations	65 (35)	
Case B	Requirements	35	Type 1*	5	1.000 (perfect agreement)	Concepts	60 (5)	161
	Conditions	1				Attributes	7 (0)	
	Constraints	18	Type 2*	0		Associations	58 (1)	
	Objects	2				Generalizations	6 (6)	
Case C	Requirements	35	Type 1*	0	0.850 (strong agreement)	Concepts	54 (16)	312
	Conditions	8				Attributes	17 (0)	
	Constraints	27	Type 2*	6		Associations	121 (6)	
	Objects	0				Generalizations	15 (14)	

* Type 1 refers to interdependencies where at least one out of a given set of omittable segments has to be retained. Type 2 refers to interdependencies where the removal of one omittable segment entails the removal of another omittable segment (See Section III-C2)

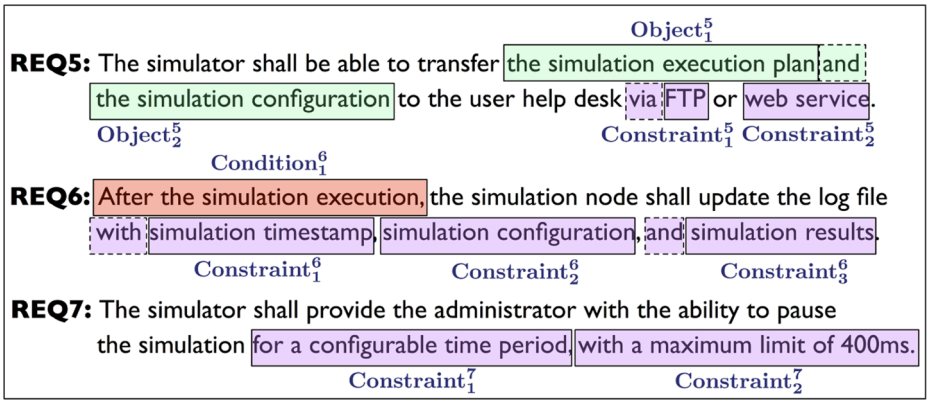

Fig. 3 Examples of omittable requirement segments

In our case studies, all omittable segments fall into one of the following types: Object, Condition, or Constraint. Objects are, in principle, omittable only when several of them are present. For example, in REQ5 there are two omittable segments of type Object: Object$_1^5$ and Object$_2^5$. It is plausible that one of the two is omitted, as long as the other one stays. In general, omitting objects is subject to the restriction that *at least one* object should remain. With regard to omittable segments of type Condition, we did not observe any interdependencies in our case studies. This is not meant to suggest that conditions with interdependencies can never exist. We just happened to have no instances in our case studies.

For constraints, we observed three possibilities in our case studies. Let $C = \{C_1, \cdots, C_n\}$ be the set of omittable segments of type Constraint in a given requirement:

(a) There are no interdependencies between C_i: Any subset of C (including the whole C) can thus be omitted. For example, in REQ5 of Fig. 3, Constraint$_1^5$ and Constraint$_2^5$ can be omitted independently, potentially both at the same time.

(b) At least one C_i needs to be retained: This is similar to what was said earlier about objects. For example, in REQ6 of Fig. 3, it is plausible that one or two out of the three constraints Constraint$_1^6$, Constraint$_2^6$ and Constraint$_3^6$ can be omitted. However, removing all three at the same time is not possible, because doing so, according to the experts, would render the remainder of the requirement evidently incomplete.

(c) If C_i is removed then C_j has to be removed as well. This typically happens when C_j is a subordinate of C_i. For example, in REQ7 of Fig. 3, removing Constraint$_1^7$ implies that Constraint$_2^7$ has to be removed at the same time.

In our case studies, we did not observe situations where a segment of one type, e.g., Object, would be related to a segment of another type, e.g., Constraint. Our simulation analysis accounts for all identified interdependencies to ensure that the seeded omissions are as realistic as possible.

3.3.3 Tracing Requirements to the Domain Model

In the last step of data collection, we had the experts in each case study trace the selected requirements, denoted $\mathcal{R}$ earlier, to the respective domain model. Recall from Section 3.3.1 that each domain model was built around the set of concepts derived from $\mathcal{R}$. This set of

concepts was earlier denoted $\mathcal{S}$. Traceability from $\mathcal{R}$ to $\mathcal{S}$ is automatic and already constitutes a large fraction of all the traceability links. What the experts were tasked to do was to identify whether and how each requirement in $\mathcal{R}$ was related to the remaining domain model elements. We illustrate this using the example of Fig. 4. Here, REQ2 from Fig. 1 is being traced to the domain model. Traceability to $\mathcal{S} = \{Simulator, Simulation, Administrator\}$ is implied. What the expert provides in addition is that "the ability to dynamically reconfigure" in REQ2 maps on to the domain model association labeled *modifies*. Establishing this link requires expertise, as only an expert can conclusively say whether dynamic reconfiguration is, at a conceptual level, a modification. In total, tracing REQ2 to the domain model yields five trace links, noting that the term "simulation" appears twice in the requirement.

Once the constituents of the requirements in $\mathcal{R}$ have been traced to the domain model as illustrated in Fig. 4, traceability from the omittable segments of these requirements to the domain model can be derived automatically on demand. This is done by taking the union of all trace links originating from a given omittable segment. In our example of Fig. 4, REQ2 has a condition, Condition_1^2. The term "simulation" in Condition_1^2 is traceable to the domain model. This induces a trace link from Condition_1^2 to *Simulation*. We note that the traceability relation from $\mathcal{R}$ to the domain model is *not* surjective (onto), meaning that not all the elements in the domain model have some corresponding text segment in the requirements. For example, no requirement is traced to *Communication Interface* and *User*, since these abstractions are tacit in the requirements.

3.3.4 Gathering Complementary Data for Analyzing Representativeness

As noted earlier in this section, our case studies consider only a fraction ($\mathcal{R}$) of the underlying documents. To be able to examine whether our results are representative of the documents in their entirety, we needed to collect additional data from the full requirements documents. This additional data collection step, which took place after the data collection activities described in Sections 3.3.1, 3.3.2 and 3.3.3, is composed of two parts: First, for each case study, two researchers (first two authors) annotated all the omittable segments of the remaining requirements (i.e., the requirements outside $\mathcal{R}$) in exactly the same manner as described in Section 3.3.2. Subsequently, interrater agreement was measured between

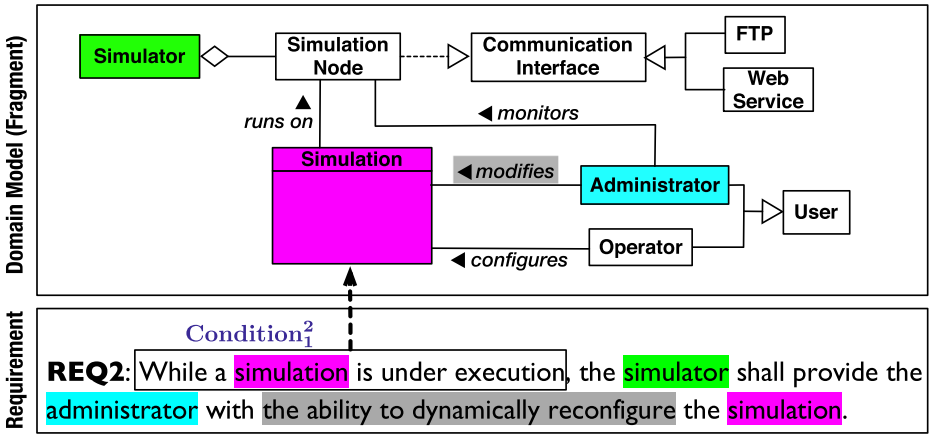

Fig. 4 Examples of trace links from a requirement to the domain model

the two sets of independently derived annotations for the full requirements documents (see Table 2). Next, and leveraging the domain knowledge gained from prior interactions with the experts (discussed in Section 3.3.2), the two researchers collaboratively identified the interdependencies between the omittable segments for the requirements outside $\mathcal{R}$. In any situation where there was doubt about the interdependencies, the experts were consulted for clarification and validation.

Finally, all the keyphrases, i.e., Noun Phrases (NPs) and Verb Phrases (VPs), were marked up in the requirements. NPs and VPs provide near-complete coverage of the meaning-bearing parts of the requirements, noting that NPs and VPs are the two major sentence constituents in many languages including English (Krzeszowski 2011). We extracted the keyphrases automatically using an NLP technology known as text chunking (Ramshaw and Marcus 1995; Jurafsky and Martin 2008). The chunker we used, Apache OpenNLP Chunker (2018), has been demonstrated to be highly accurate over requirements documents (Arora et al. 2015). Nevertheless and for quality assurance reasons, the first author examined all the automatically extracted keyphrases, and where necessary, made slight corrections.

3.4 Analysis Procedure

Our analysis is a form of Monte-Carlo simulation (Robert and Casella 2005). We present our simulation algorithm in Fig 5. The inputs to the algorithm are: (1) the outcomes of our data collection procedure, described in Section 3.3, (2) the type of omissions to simulate (T), and (3) the number of simulation runs (n).

Following a randomized process, the algorithm seeds into requirements progressively larger sets of omissions of a given type (L. 5-6 of the algorithm), e.g., omissions of constraints ($T = \texttt{Cons}$) or whole requirements ($T = \texttt{Req}$).

As noted earlier, for omissions of type Object and Constraint, one has to consider the interdependencies between the omittable segments. A randomly-selected set of omittable segments, denoted $\mathcal{E}$ in the algorithm (L. 6), may violate the interdependencies. Such violations need to be resolved by making modifications to $\mathcal{E}$ (L. 7-8). Noting the simple nature of the interdependencies in our case studies (Section 3.3.2), we can resolve any violations in $\mathcal{E}$ through a simple algorithm (not discussed), without using search-based or backtracking solvers.

Next, we identify the domain model elements which are supported by some trace link *before* the selected omissions are applied, but *not after* the omissions are applied (L. 9-11). This gives us the set $\mathcal{U}$. The loss of support for the elements in $\mathcal{U}$ may also have implications for the domain model elements that are tacit in the requirements. For example, suppose that $\mathcal{U}$ contains both *Administrator* and *Operator* from the model fragment of Fig. 1b; this happens when $\mathcal{E}$ contains REQ1, REQ2, and REQ3 of Fig. 1a. With *Administrator* and *Operator* no longer supported, the abstraction of *User* loses its support too. This implied loss of support for the domain model elements that are tacit in the requirements is captured by the set $\mathcal{X}$ (L. 12).

What is recorded in the output (scatter plot) at the end of each iteration of the loop of L. 5 is a tuple. The first value in the tuple is the number of omissions seeded, i.e, $|\mathcal{E}|$; the second value is the percentage of domain model elements that have lost their support either directly or indirectly, i.e., $|\mathcal{U}| + |\mathcal{X}|$ normalized by the total number of domain model elements. The sensitivity of the domain model to omissions is measured by how quickly the percentage of unsupported domain model elements grows as increasingly larger sets of omissions are

Table 2 Results of complementary data collection for the full requirements documents

Case	# of omittable parts		# of interdependencies		Interrater agreement (Cohen's κ)	# of keyphrases (NPs, VPs)
Case A	Requirements	163	Type 1*	29	0.939 (almost perfect agreement)	515 (380, 135)
	Conditions	12				
	Constraints	107	Type 2*	2		
	Objects	86				
Case B	Requirements	212	Type 1*	17	1.000 (perfect agreement)	615 (419, 196)
	Conditions	5				
	Constraints	53	Type 2*	0		
	Objects	5				
Case C	Requirements	110	Type 1*	0	0.896 (strong agreement)	189 (98, 91)
	Conditions	34				
	Constraints	56	Type 2*	18		
	Objects	0				

* Type 1 and Type 2 are defined as in Table 1.

applied. Since the loop of L. 5 has a random component, it has to be run multiple times to account for random variation (L. 4). The scatter plot resulting from running the algorithm of Fig. 5 is the basis for answering RQ1 in Section 4.

4 Results and Discussion

In this section, we (i) discuss the results of our case studies, (ii) answer the research questions posed in Section 3.1, and (iii) argue about the representativeness of the requirements samples used in our case studies.

Table 1 provides key statistics about the outcomes of our data collection as per the procedures discussed in Sections 3.3.1, 3.3.2 and 3.3.3. For each case study, the table provides the following information: (1) the number of omittable elements of different types: as explained in Section 3.3.2, an omittable element can be an entire requirement or a certain segment of a requirement, namely a condition, constraint, or object; (2) the number and type of interdependencies between the omittable constraints and objects (the types were discussed in Section 3.3.2); (3) interrater agreement, computed as Cohen's κ (1960), for the identification and classification of omittable segments by two coders. The κ scores indicate strong, almost perfect or perfect agreement in all case studies; (4) the number of domain model elements (of which the number of elements tacit in the requirements is shown in brackets); and (5) the number of trace links from the requirements to the domain model.

Algorithm Simulation

Input: - A domain model
- Set of requirements, their omittable segments, and the interdependencies between these segments
- Set $\mathcal{L}$ of trace links from the requirements to the domain model
- Omission type $T \in \{\texttt{Req}, \texttt{Cond}, \texttt{Cons}, \texttt{Obj}\}$ to simulate
- Number n of simulation runs

Output: - Scatter plot Plot showing the percentage of unsupported domain model elements versus the number of omissions

1. Let $\mathcal{K}$ be the set of domain model elements *not* supported by $\mathcal{L}$
2. Let $\mathcal{O}$ be the set of all omittable segments of type T
3. Let Plot be initially empty
4. **for** $i = 1$ **to** n:
5. **for** $j = 1$ **to** $|\mathcal{O}|$:
6. Randomly pick j elements, $\mathcal{E} = \{e_1, \cdots, e_j\}$, from $\mathcal{O}$
7. **if** ($T = \texttt{Cons}$ or $T = \texttt{Obj}$)
8. Minimally modify $\mathcal{E}$ to resolve any interdependency violations
9. Let $\mathcal{L}^-$ be the set of all trace links originating from $\mathcal{E}$
10. $\mathcal{L}' = \mathcal{L} \setminus \mathcal{L}^-$
11. Let $\mathcal{U}$ be the set of domain model elements supported by $\mathcal{L}$ but *not* $\mathcal{L}'$
12. Let $\mathcal{X}$ be the domain elements in $\mathcal{K}$ that will become unreachable from non-$\mathcal{K}$ elements if $\mathcal{U}$ is removed from the domain model
13. Place the following datapoint on Plot: $(|\mathcal{E}|, (|\mathcal{U}| + |\mathcal{X}|)/D)$, where D is the total number of domain model elements
14. **return** Plot

Fig. 5 Simulation algorithm for computing sensitivity

Table 2 provides statistics about the complementary data that we gathered according to the procedure described in Section 3.3.4 and with the goal of examining the representativeness of our case studies. Specifically, Table 2 provides the following for each case study: the same information as in Table 1 under (1), (2), and (3), but now for the full requirements documents. The κ scores over the full documents indicate strong, almost perfect or perfect agreement; and (4) the number of keyphrases (NPs and VPs) in the full documents.

The amount of time the researchers spent with the experts for data collection is ≈8 hours for each Case A and Case B, and ≈6 hours for Case C. These numbers should be interpreted with caution as they do not represent the full effort invested by the experts; we recall from Section 3.3.1 that the experts (1) did not create the domain models from scratch but rather based their work on existing domain-model sketches, and (2) did parts of the domain modeling offline without the researchers being present.

RQ1. Figure 6 shows the output (scatter plots) for $n = 100$ runs of our simulation algorithm (Fig. 5). Each plot shows, for each case study, the relationship between the number of omissions of a certain type (x-axis) and the percentage of domain model elements that are no longer supported as the result of the omissions (y-axis). The plots further show, for each case study, the mean curve of the 100 runs. Before presenting our conclusions from these plots, we note that in the plots of Fig. 6c and d, the maximum number of omitted constraints and objects, respectively, is less than the total number of these two omittable types in Table 1. For instance, in the plot of Fig. 6c, the maximum number of constraints omitted in

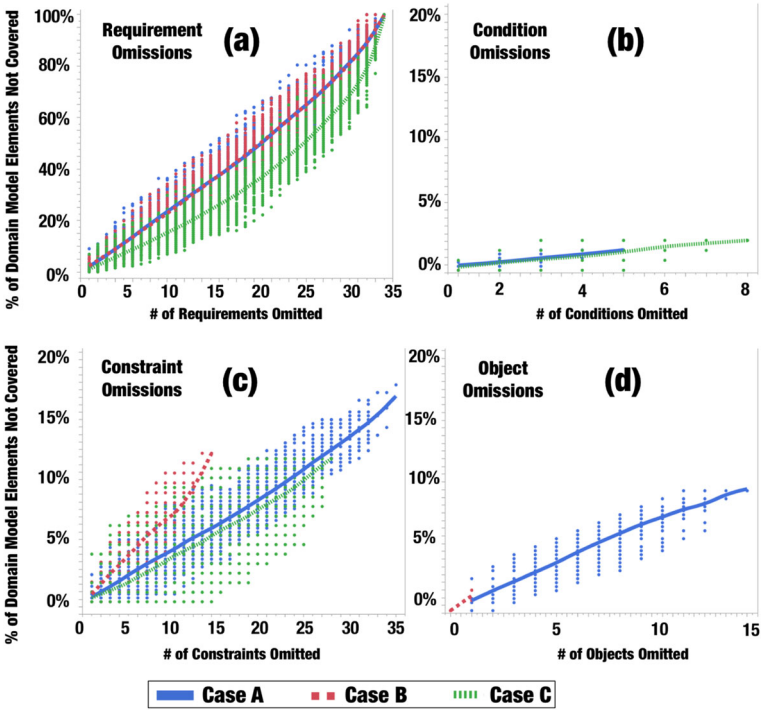

Fig. 6 Plots showing the relationship between unsupported domain model elements and number of omissions of different types

Case A is 34, although the total number of constraints marked as omittable in this case study is 42. This, we recall from our discussion of Section 3.3.2, is due to the interdependencies between segments of these two types.

From Fig. 6a, we observe that the mean curves for Case A and Case B show a virtually linear increase in the percentage of domain model elements that lose support (i.e., are no longer supported) as more requirements are omitted in their entirety. In Case C, the level of sensitivity to the omission of requirements is still high, but the sensitivity is visibly lower than in Case A and Case B. Stated otherwise, it takes more omissions in Case C than in Case A and Case B for the domain model elements to lose their support.

A number of influencing factors may be at play, distinguishing Case C from the other two cases. While a thorough examination of these factors is difficult and requires more case studies, we observe one important distinction between Case C and the other two. In Section 1, we argued that the frequency of appearance of domain concepts in the requirements has an impact on the sensitivity of domain model to omissions. In Case C, the domain concepts appear considerably more frequently in the requirements than in Case A and Case B. The mean frequency of appearance of domain concepts in the requirements, – in other words, the average number of trace links from the requirements or segments thereof to the domain concepts – in Case C is 7.4 (SD = 10.9); this frequency is 1.5 (SD = 1.6) in Case A and 1.7 (SD = 2.6) in Case B. Consequently, in Case C, when compared to the other two cases, the domain concepts on average have more support in the requirements. This partly explains why Case C has lower sensitivity to the omission of requirements.

Figure 6b, c and d show the sensitivity of the domain models in our case studies to omissions of conditions, constraints, and objects, respectively. The slopes of the mean curves are not comparable across the different plots in Fig. 6 due to the axes of the plots having different scales. We compare the slopes of the mean curves using Fig. 7. For each case

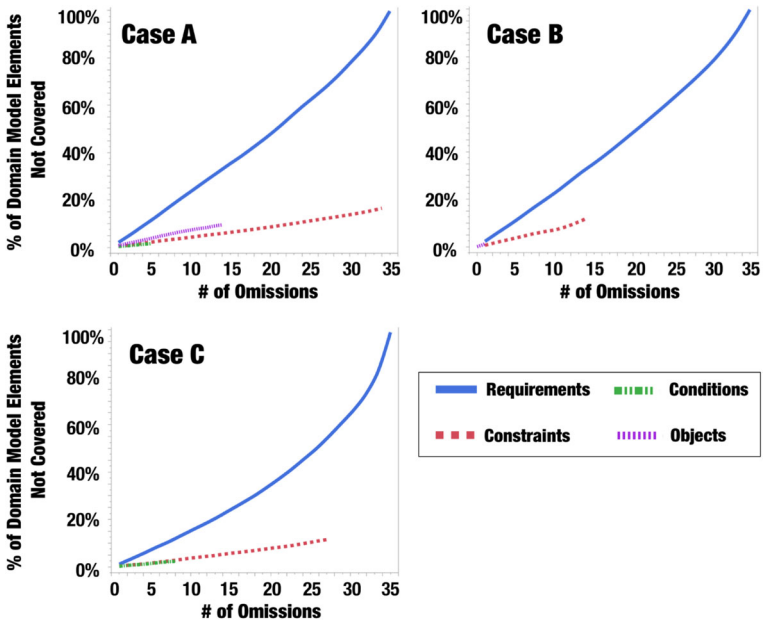

Fig. 7 Sensitivity of domain models to different types of omissions in NL requirements

study, this figure shows how quickly domain model elements lose support as the number of omissions of different types increases. In all case studies, the omission of whole requirements has a much larger impact than the omission of requirement segments. For instance, in Case A, the omission of five requirements on average leaves 12.5% of the domain model elements unsupported. Omitting the same number of conditions, constraints, and objects on average leaves only 1.5%, 2.3% and 4% of the domain model elements unsupported, respectively. More precisely, the level of sensitivity, as indicated by the slope of a linear function, is on average 4.4 times higher for requirement omissions than for the other three omission types considered. This suggests that domain models are more sensitive to unspecified requirements, i.e., requirements that are missing, than under-specified requirements, i.e., requirements whose details are incomplete. One explanation for this phenomenon is that the non-omittable segments are more essential than the omittable ones to the meaning of the requirements. Consequently, domain concepts, which are, by definition, core to a given domain and thus to the requirements specified within that domain, are more likely to find their place in the non-omittable segments. A complementary explanation is that, compared to the omittable segments, the non-omittable segments constitute a larger proportion of the overall content of the requirements, thus making the non-omittable segments responsible for a larger proportion of the overlap between the domain model and the requirements. For example, in Fig. 4, Condition_1^2 (the only omittable segment in the underlying requirement) is traced to one domain model element, whereas the non-omittable segments are traced to four domain model elements.

We make the following further remarks about the plots in Figs. 6 and 7. In Case B, (1) there is only one condition which, when omitted, has no impact, (2) there are two omittable objects, out of which at most one can be removed in any given simulation run. The average impact for an object omission in Case B is 1.5%. In Case C, (1) there are no omittable objects, (2) if omitted in equal numbers, the impact of constraint omissions and condition omissions is almost the same, as suggested by the slopes of the curves in Fig. 7 for Case C.

*The answer to **RQ1** is that domain models show near-linear sensitivity to omissions in requirements. The level of sensitivity is, on average, 4.4 times higher when requirements are removed in their entirety than when conditions, constraints and objects are removed.*

RQ2. As noted in RQ1, there is variation across our case studies in terms of how sensitive domain models are to the omission of requirements. For example, the impact of omitting 10 requirements from Case A is the loss of support for 23.4% of the domain model elements. In comparison, this number is 22.7% for Case B and 15.3% for Case C. A natural question that arises here is the following: In a real setting, how can one know the level of sensitivity of a domain model to different omissions? When a domain model is already existing, answering the above question helps the analysts gauge the utility of the model for detecting incompleteness in requirements. When a domain model is non-existing, answering this question is one (but not the only) important parameter in deciding whether a domain model is worthwhile building.

Obviously, outside an evaluation setting, one would not have a gold standard to perform the same type of analysis as in RQ1. The goal should therefore be to somehow predict sensitivity based on the *intrinsic characteristics* of a requirements document. A useful such characteristic, implied by our discussion in RQ1, is the frequency of terms. However, this is only one of potentially many influencing factors. For example, how quickly the support for

the associations of a domain model is lost is more likely to be influenced by the frequency of pairs of terms collocated in the same sentence, rather than the frequency of individual terms.

Our current study does not allow us to systematically examine the factors that influence the sensitivity of domain models to omissions in requirements. In lieu of such an examination, we have developed a useful surrogate for predicting sensitivity, which we present next: we propose to run the simulation algorithm of Fig. 5 with the y-axis replaced with the percentage of unsupported *keyphrases (NPs and VPs)* as opposed to the percentage of unsupported domain model elements, which is not measurable outside an evaluation setting. The idea of using unsupported keyphrases as a surrogate comes from the observation that NPs and VPs are the main meaning-carrying units of requirements statements (Arora et al. 2017). Determining whether a keyphrase is still supported after a set of omissions is trivial: the phrase either still appears (is supported) in the document after the omissions, or it no longer does (is unsupported).

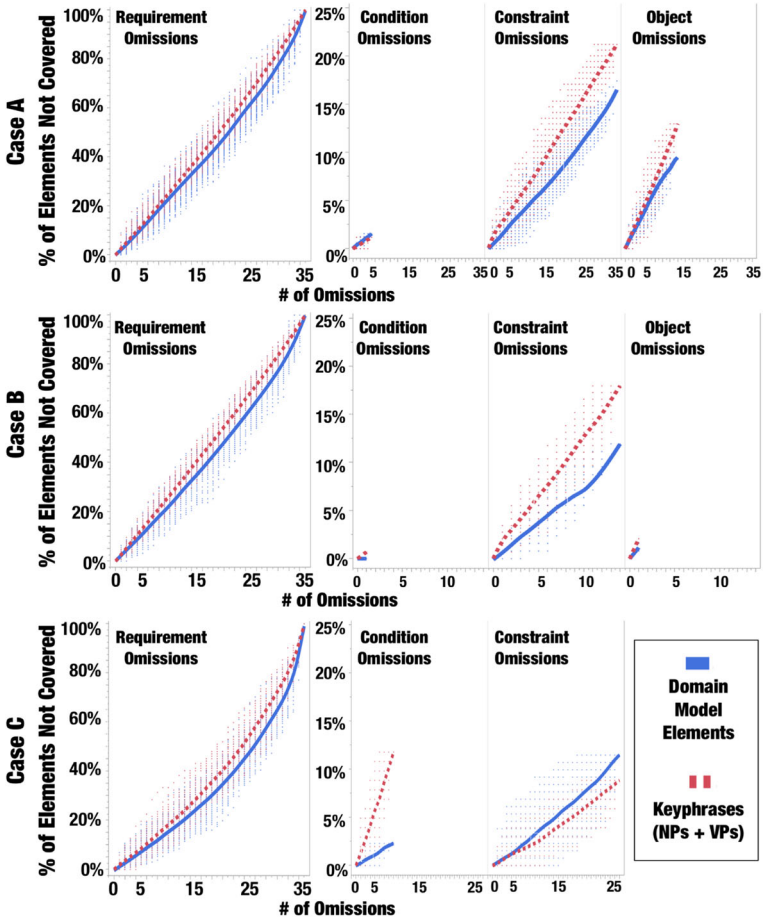

Fig. 8 Using unsupported keyphrases as a surrogate for predicting sensitivity

Figure 8 plots for each case study unsupported keyphrases against unsupported domain model elements, organized by omission type. As in RQ1, the plots have been generated by 100 runs of our simulation algorithm in Fig. 5. We note that the y-axis for requirement omissions has a range of 0–100%, whereas the range for other omission types is 0–25%. Generally speaking, the mean curves for unsupported keyphrases and unsupported domain model elements follow the same trend, taking into account the fact that the curves for omissions of type Condition and Object cannot be reliability compared due to the low prevalence of these omission types across our case studies.

Using keyphrases as a surrogate for domain model elements is most practical for predicting sensitivity to requirement omissions. Although the results of Fig. 8 suggest that keyphrases yield a good sensitivity predictor for constraint omissions as well, building such a predictor is not very practical. This is because doing so would require the analysts to first delineate all the constraints in a given requirements document.

To precisely analyze the curves in Fig. 8 obtained for whole requirement omissions, we computed the correlation between the percentage of unsupported keyphrases and the percentage of unsupported domain model elements, using the datapoints resulting from the 100 simulation runs. The Pearson's correlation coefficient (Warner 2012) for Case A is 0.97440 ($p < 0.0001$), for Case B is 0.98005 ($p < 0.0001$), and for Case C is 0.96838 ($p < 0.0001$).

*The answer to **RQ2** is that to predict domain model sensitivity to the omission of requirements, one can run the simulation algorithm of Fig. 5 by replacing the y-axis with the percentage of unsupported NPs and VPs.*

Representativeness. As discussed in Section 3, each of our case studies considers only a subset (35 requirements) of the underlying requirements document. It is thus important to examine whether our analysis of RQ1 would be representative of the full requirements documents as well.

We argue about representativeness indirectly and through RQ2. In particular, we know from RQ2 that, keyphrases are a good proxy for measuring the sensitivity of domain model elements to omissions. If the keyphrase characteristics of the requirements subsets in our case studies happen to be close to those of the full requirements documents, we can conclude that the subsets have not drifted too far from the original documents in terms of their characteristics. In other words, if we were to answer RQ1 using the full requirements documents, we would, with reasonable confidence, obtain similar results to those obtained over the subsets.

Figure 9 shows the keyphrase sensitivity curves for both the requirements subsets and the full documents, organized by case study. The charts in this figure have been computed using the same method as those in Fig. 8. The only difference is that the x-axis in the charts of Fig. 9 has been normalized to enable comparing, on a common scale, the requirements subsets and the full documents. The omittable parts required for applying our simulation algorithm to the full requirements documents resulted from the data collection step in Section 3.3.4. Statistics about the identified omittable parts were already provided in Table 2.

As seen from Fig. 9, while the characteristics of the full requirements documents are not exactly the same as those of the requirements subsets in our case studies, there are no remarkable discrepancies between the characteristics. We note that, just like in Fig. 8, the

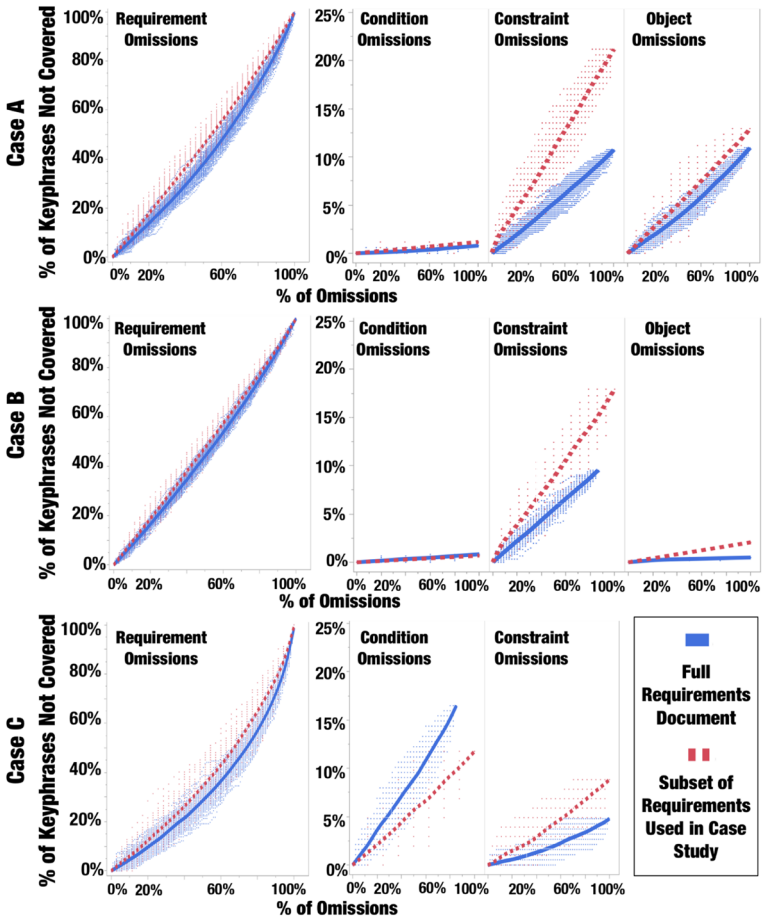

Fig. 9 Comparison of the sensitivity of keyphrases to different omission types in the full requirements documents versus the requirements subsets used in our case studies

y-axis in Fig. 9 has a range of 0–100% for requirements omissions and a range of 0–25% for the other omission types.

> *In light of RQ2, and considering that the keyphrase characteristics of the requirements in our case studies are not far from those of the full documents the requirements were drawn from, we conclude that our case studies are reasonably representative of the full requirements documents. This provides some degree of confidence that the sensitivity levels observed in RQ1 would generalize to the full requirements documents.*

5 Threats to Validity

Internal Validity. The main threat to the internal validity of our work is subjectivity in modeling. This applies both to *what* the modelers include in their domain models (subjectivity

about content), and also to *how* the modelers express the desired content (subjectivity about form). To minimize subjectivity about content, we engaged closely with at least two domain experts in each case study. These experts had an in-depth understanding of their respective application domains, and made reasoned choices about what belonged in a domain model. We believe that other experts, under identical circumstances, would have made similar choices about content. The potential influence of subjectivity about content is further mitigated by the fact that we obtain consistent results across multiple case studies: if distinct case studies yield the same outcome, repeating any one of them with different experts would be unlikely to affect the findings.

With regard to subjectivity about form, we observe that such subjectivity is common in domain models represented as class diagrams (Larman 2004). As a simple example, what one expert might find to be a class may be deemed as an attribute by another expert. To mitigate subjectivity about form, we made our analysis independent of domain model element types. In other words, we do not distinguish classes, attributes, associations and generalizations when analyzing sensitivity; each element, regardless of its type, counts as one information element.

Conclusion Validity. In measuring sensitivity, we are concerned only with whether domain models contain the information that is necessary for revealing omissions in requirements. As we illustrated in our example of Section 1, a domain model may contain information beyond what one would normally express within the requirements. We denoted such information by D_{only} (see page 5). When one systematically inspects the elements of a domain model in order to identify whether these elements have counterparts in the requirements, any element that happens to be in D_{only} will raise a false alarm, i.e., a warning about information that has been deliberately left tacit in the requirements. Our study is not meant at measuring the overhead associated with investigating such false alarms. We thus cannot reach conclusions about the cost-benefit tradeoffs of using domain models for completeness checking. We do note however that, since detecting incompleteness is generally a very difficult problem, the above overhead appears to be a reasonable price to pay for the genuine requirements incompleteness issues that one can identify through a domain model.

Construct Validity. It is paramount to point out that our constructs do not measure whether analysts actually detect an omission in the requirements when the domain model holds the right cues about the omission. Stated otherwise, there is no guarantee that human inspectors would spot an omission even when our analysis indicates they should. What our current evaluation measures is the *potential* of domain models for revealing omissions in the requirements. To what extent analysts can exploit this potential requires further investigation and is outside the scope of our current work.

Another consideration related to construct validity is that differences may exist between the terminology in the requirements and that in the domain model. Such discrepancies, if ignored, can have a confounding effect on the sensitivity analysis of RQ1. In practice, if a domain model is to be employed for completeness checking of requirements, one needs to either reconcile terminological variations in the requirements, or group together the synonym terms. In our evaluation, we mitigated this threat by ensuring that, in each case study, at least one of the requirements authors was participating. The variations in terminology were duly accounted for while establishing traceability from the requirements to the domain model.

External Validity. Our results are confirmed via three distinct case studies, thus providing confidence about the external validity of our findings. That being said, we emphasize that our findings are based upon the assumption that the domain models are reasonably complete. Although built by experts, the domain models in our case studies were scoped to a fraction of the underlying application domains, so that the experts could take the necessary time to be thorough in their elaboration. In a non-evaluation setting, the completeness of a domain model will inevitably be affected by factors such as time and budget constraints, which our current studies were not subject to. What our results suggest is that, if built with adequate diligence, domain models are useful instruments for completeness checking of requirements. Whether one would get similar benefits from domain models constructed "in the wild" requires further investigation.

6 Conclusion

Domain models are typically used as an apparatus for defining a consistent development terminology and improving communication between different stakeholders. Our work in this article was motivated by the anecdotal observation that domain models are additionally helpful for detecting incompleteness in requirements. To empirically examine this observation, we conducted three industrial case studies with subject-matter experts. In each case study, we measured the incompleteness-revealing-power of a domain model by seeding realistic omissions into the requirements, and determining whether the domain model held information that would logically signal the presence of the omissions.

Our results provide empirical evidence that domain models can indeed provide useful hints toward the detection of incompleteness in textual requirements. As pointed out throughout the article, our work focused on domain models built in ideal conditions. A natural followup to our current work would be to investigate whether domain models built under time and cost pressures would offer comparable benefits. Another important future avenue of work is to conduct user studies aimed at assessing how effectively analysts can exploit a domain model for finding incompleteness issues in requirements. To be able to conclusively measure the relative usefulness of domain models for detecting such issues, we first need to establish a baseline by confronting subject-matter experts with requirements from which some segments have been artificially removed and having them mark the requirements that are missing information. Finally, in this article, we addressed the issue of completeness checking solely from the lens of functional requirements. It would be interesting to study whether models with the capacity to capture non-functional aspects, e.g., goal models, can be utilized in a similar manner for uncovering incompleteness in textual descriptions of non-functional requirements.

Acknowledgments This project has received funding from the Luxembourg National Research Fund (FNR) under grants FNR/P10/03 and FNR-11601446, and from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 694277).

References

Alrajeh D, Kramer J, Van Lamsweerde A, Russo A, Uchitel S (2012) Generating obstacle conditions for requirements completeness. In: Proceedings of the 34th international conference on software engineering (ICSE'12), pp 705–715

Ambler S (2004) The object primer: agile model-driven development with UML 2.0, 3rd edn. Cambridge University Press, Cambridge

Apache OpenNLP Chunker (2018) Apache OpenNLP Chunker. http://opennlp.apache.org, last accessed: November 2018

Arora C, Sabetzadeh M, Briand L, Zimmer F (2015) Automated checking of conformance to requirements templates using natural language processing. IEEE Trans Softw Eng 41(10)

Arora C, Sabetzadeh M, Briand L, Zimmer F (2017) Automated extraction and clustering of requirements glossary terms. IEEE Trans Software Eng (IEEE TSE) 43(10):918–945

Basili VR, Weiss DM (1981) Evaluation of a software requirements document by analysis of change data. In: Proceedings of the 5th international conference on software engineering (ICSE'81), pp 314–323

Basili VR, Green S, Laitenberger O, Lanubile F, Shull F, Sørumgård S, Zelkowitz MV (1996) The empirical investigation of perspective-based reading. Empir Softw Eng 1(2):133–164

Boehm B (1984) Verifying and validating software requirements and design specifications. IEEE Softw 1(1):75–88

Bravoco RR, Yadav SB (1985) A methodology to model the functional structure of an organization. Comput Ind 6(5):345–361

Chimiak-Opoka J (2009) OCLLib, OCLUnit, OCLDoc: pragmatic extensions for the object constraint language. In: Proceedings of 12th international conference on model driven engineering languages and systems (MODELS'09). Springer, pp 665–669

Cockburn A (1997) Structuring use cases with goals. Journal of Object-Oriented Programming 10(5):56–62

Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20(1):37–46

Constantine LL, Lockwood LAD (1999) Software for use: a practical guide to the models and methods of usage-centered design. ACM Press/Addison-Wesley Publishing Co, New York

Costello RJ, Liu DB (1995) Metrics for requirements engineering. J Syst Softw 29(1):39–63

Dalpiaz F, van der Schalk I, Lucassen G (2018) Pinpointing ambiguity and incompleteness in RE via information visualization and NLP. In: Proceedings of the 24th international working conference on requirements engineering: foundation for software quality, (REFSQ'18)

Davis A, Overmyer S, Jordan K, Caruso J, Dandashi F, Dinh A, Kincaid G, Ledeboer G, Reynolds P, Sitaram P, Ta A, Theofanos M (1993) Identifying and measuring quality in a software requirements specification. In: Proceedings of the 1st international software metrics symposium (METRICS'93), pp 141–152

Davis AM (1990) Software requirements: analysis and specification. Prentice Hall Press, Upper Saddle River

Easterbrook SM, Yu ESK, Aranda J, Fan Y, Horkoff J, Leica M, Qadir RA (2005) Do viewpoints lead to better conceptual models? an exploratory case study. In: Proceedings of the 13th international conference on requirements engineering (RE'05), pp 199–208

Eckhardt J, Vogelsang A, Femmer H, Mager P (2016) Challenging incompleteness of performance requirements by sentence patterns. In: Proceedings of the 24th international requirements engineering conference (RE'16). IEEE, pp 46–55

España S, Condori-Fernandez N, Gonzalez A, Pastor Ó (2009) Evaluating the completeness and granularity of functional requirements specifications: a controlled experiment. In: Proccedings of the 17th international requirements engineering conference, (RE'09), pp 161–170

España S, González A, Pastor Ó (2009) Communication analysis: a requirements elicitation approach for information systems. In: Proceedings of the 21st international conference on advanced information systems (CAiSE'09), pp 530–545

Evans E (2004) Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional, Reading

Ferrari A, Dell'Orletta F, Spagnolo GO, Gnesi S (2014) Measuring and improving the completeness of natural language requirements. In: Proceedings of the 20th international working conference on requirements engineering: foundation for software quality (REFSQ'14), pp 23–38

Fusaro P, Lanubile F, Visaggio G (1997) A replicated experiment to assess requirements inspection techniques. Empir Softw Eng 2(1):39–57

Gane CP, Sarson T (1979) Structured systems analysis: tools and techniques. Prentice Hall Professional, Upper Saddle River

Geierhos M, Bäumer FS (2016) How to complete customer requirements. In: Proceedings of the 21st international conference on natural language processing and information systems (NLDB'16), pp 37–47

Gigante G, Gargiulo F, Ficco M (2015) A semantic driven approach for requirements verification. In: Intelligent distributed computing VIII. Springer, pp 427–436
Heimdahl MPE, Leveson NG (1996) Completeness and consistency in hierarchical state-based requirements. IEEE Trans Softw Eng 22(6):363–377
Heitmeyer C, Kirby J, Labaw B (610) The SCR method for formally specifying, verifying, and validating requirements: tool support. In: Proceedings of the 19th international conference on software engineering (ICSE'97). ACM
Holt J, Perry S, Brownsword M (2011) Model-based requirements engineering. IET
IEEE (2011) IEEE 29148:2011 - systems and software engineering-requirements engineering
Jaffe MS, Leveson NG, Heimdahl MPE, Melhart BE (1991) Software requirements analysis for real-time process-control systems. IEEE Trans Softw Eng 17(3):241–258
Jurafsky D, Martin J (2008) Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd edn. Prentice Hall, Upper Saddle River
Kaiya H, Saeki M (2005) Ontology based requirements analysis: lightweight semantic processing approach. In: Proceedings of the 5th international conference on quality software (QSIC'05), pp 223–230
Kamalrudin M, Hosking J, Grundy J (2011) Improving requirements quality using essential use case interaction patterns. In: Proceedings of the 33rd international conference on software engineering (ICSE'11), pp 531–540
Khatwani C, Jin X, Niu N, Koshoffer A, Newman L, Savolainen J (2017) Advancing viewpoint merging in requirements engineering: a theoretical replication and explanatory study. Requir Eng 22(3):317–338
Koenig T, Olsson T, Schmid K, Adam S (2007) Influence of requirements specification notation on change impact analysis: an empirical investigation. Tech rep, Fraunhofer IESE, Kaiserslautern
Krzeszowski TP (2011) Contrasting languages: the scope of contrastive linguistics, vol 51. Walter de Gruyter, Berlin
Larkin JH, Simon HA (1987) Why a diagram is (sometimes) worth ten thousand words. Cognit Sci 11(1):65–100
Larman C (2004) Applying UML and Patterns: an introduction to object-oriented analysis and design and iterative development, 3rd edn. Prentice Hall, Upper Saddle River
Lindland OI, Sindre G, Solvberg A (1994) Understanding quality in conceptual modeling. IEEE Softw 11(2):42–49
Menzel I, Mueller M, Gross A, Doerr J (2010) An experimental comparison regarding the completeness of functional requirements specifications. In: Proceedings of the 18th international requirements engineering conference (RE'10), pp 15–24
Mich L, Franch M, Pierluigi NI (2004) Market research for requirements analysis using linguistic tools. Requirements Engineering Journal 9(1):40–56
Moody DL, Shanks GG, Darke P (1998) Improving the quality of entity relationship models—experience in research and practice. In: Conceptual modeling – Proceedings of the 17th international conference on conceptual modeling (ER'98). Springer, pp 255–276
Nuseibeh B, Kramer J, Finkelstein A (1994) A framework for expressing the relationships between multiple views in requirements specification. IEEE Trans Softw Eng 20(10):760–773
Object Management Group (2004) Object constraint language 2.4 specification. http://www.omg.org/spec/OCL/2.4/, last accessed: Feb. 2018
Pohl K (1993) The three dimensions of requirements engineering. In: Proceedings of the 5th international conference on advanced information systems engineering (CAiSE'93), pp 275–292
Pohl K, Rupp C (2011) Requirements engineering fundamentals, 1st edn. Rocky Nook, San Rafael
Porter A, Votta L (1998) Comparing detection methods for software requirements inspections: a replication using professional subjects. Empir Softw Eng 3(4):355–379
Ramshaw L, Marcus M (1995) Text chunking using transformation-based learning. In: 3rd ACL workshop on very large corpora, ACL, pp 82–94
Reggio G, Leotta M, Ricca F (2014) Who knows/uses what of the UML: a personal opinion survey. In: Proceedings of the 17th international conference on model driven engineering languages and systems (MODELS'14), pp 149–165
Robert C, Casella G (2005) Monte Carlo statistical methods, 2nd edn. Springer, Berlin
Sabetzadeh M, Easterbrook SM (2006) View merging in the presence of incompleteness and inconsistency. Requirements Engineering Journal 11(3):174–193
Salger F, Engels G, Hofmann A (2009) Inspection effectiveness for different quality attributes of software requirement specifications: an industrial case study. In: ICSE workshop on software quality (WOSQ'09), pp 15–21

Schneider GM, Martin J, Tsai WT (1992) An experimental study of fault detection in user requirements documents. ACM Trans Softw Eng Methodol 1(2):188–204
Schuette R, Rotthowe T (1998) The guidelines of modeling – an approach to enhance the quality in information models. In: Proceedings of the 17th international conference on conceptual modeling (ER'98), pp 240–254
Thelin T, Runeson P, Wohlin C (2003) An experimental comparison of usage-based and checklist-based reading. IEEE Trans Softw Eng 29(8):687–704
W3C OWL Working Group (2012) OWL 2 web ontology language document overview, 2nd edn. W3C
Warner RM (2012) Applied statistics: from bivariate through multivariate techniques, 2nd edn. SAGE Publications, Thousand Oaks
Whittle J, Sawyer P, Bencomo N, Cheng BHC, Bruel JM (2009) RELAX: incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 17th international requirements engineering conference (RE'09). IEEE, pp 79–88
Whittle J, Hutchinson J, Rouncefield M (2014) The state of practice in model-driven engineering. IEEE Softw 31(3):79–85
Yadav SB, Bravoco RR, Chatfield AT, Rajkumar TM (1988) Comparison of analysis techniques for information requirement determination. Commun ACM 31(9):1090–1097
Zhou J, Lu Y, Lundqvist K (2014) A TASM-based requirements validation approach for safety-critical embedded systems. In: Proceedings of the 19th international conference on reliable software technologies (Ada-Europe'14), pp 43–57
Zowghi D, Gervasi V (2003a) On the interplay between consistency, completeness, and correctness in requirements evolution. Inf Softw Technol 45(14):993–1009
Zowghi D, Gervasi V (2003b) The three Cs of requirements: Consistency, completeness, and correctness. In: Proceedings of the 8th international workshop on requirements engineering: foundation for software quality, (REFSQ'02)

Chetan Arora is an Industrial Research Fellow with a dual affiliation to Interdisciplinary Centre for Security, Reliability and Trust (Luxembourg) and SES Networks (Luxembourg). He received the Ph.D. degree from the University of Luxembourg (Luxembourg) in 2016. Over the last decade, Arora has been working in applied research in collaboration with several industry and government sectors, including satellite communication, automotive and crisis management. His research interests include requirements engineering, empirical software engineering, and applied machine learning and natural language processing.

Mehrdad Sabetzadeh is a Senior Research Scientist at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. Prior to joining SnT, he worked as a permanent member of the research staff at Simula Research Laboratory (Norway), and as an NSERC postdoctoral fellow at University College London (UK). Sabetzadeh received his Ph.D. in Computer Science from the University of Toronto. His main research interests are in software engineering with an emphasis on requirements engineering, model-based development, and regulatory compliance. Sabetzadeh is passionate about fostering stronger ties between academia and industry; in the past ten years, he has conducted most of his research in close collaboration with industry partners. His experience spans several sectors, including government, finance, legal services, telecommunications, maritime, energy, aerospace, railways, and automotive. Sabetzadeh has coauthored more than 70 scientific papers and secured more than €4m of research funding as lead investigator. He has been on the organizing or program committees of several international conferences such as RE, ESEC/FSE, MODELS, and ESEM.

Lionel C. Briand is professor in software verification and validation at the SnT centre for Security, Reliability, and Trust, University of Luxembourg, where he is also the vice-director of the centre. Lionel started his career as a software engineer in France (CS Communications & Systems) and has conducted applied research in collaboration with industry for more than 25 years.

Until moving to Luxembourg in January 2012, he was heading the Certus center for software verification and validation at Simula Research Laboratory, where he was leading applied research projects in collaboration with industrial partners. Before that, he was on the faculty of the department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he was full professor and held the Canada Research Chair (Tier I) in Software Quality Engineering. He has also been the software quality engineering department head at the Fraunhofer Institute for Experimental Software Engineering, Germany, and worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA Goddard Space Flight Center, CSC, and the University of Maryland, USA.

Lionel has been on the program, steering, or organization committees of many international, IEEE and ACM conferences. He is the coeditor-in-chief of Empirical Software Engineering (Springer) and is a member of the editorial boards of Systems and Software Modeling (Springer) and Software Testing, Verification, and Reliability (Wiley). He was on the board of IEEE Transactions on Software Engineering from 2000 to 2004.

Lionel was elevated to the grade of IEEE Fellow in 2010 for his work on the testing of object-oriented systems. He was granted the IEEE Computer Society Harlan Mills award and the IEEE Reliability Society engineer-of-the-year award for his work on model-based verification and testing, respectively in 2012 and 2013. He received an ERC Advanced grant in 2016, which is the most prestigious individual research grant in the EU. His research interests include: software testing and verification, model-driven software development, search-based software engineering, and empirical software engineering.

Affiliations

Chetan Arora[1,2] · Mehrdad Sabetzadeh[1] · Lionel C. Briand[1]

Mehrdad Sabetzadeh
sabetzadeh@svv.lu

Lionel C. Briand
briand@svv.lu

[1] Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, 29 Avenue John F. Kennedy, L-1855, Luxembourg, Luxembourg

[2] SES Networks, Rue Pierre Werner, L-6815 Betzdorf, Luxembourg