

# Disambiguating Software Property Descriptions: A Solution Proposal and Evaluation

Efi Papatheocharous<sup>\*</sup>, Séverine Sentilles, Kai Petersen, Federico Ciccozzi

<sup>\*</sup>Corresponding author: [efi.papatheocharous@ri.se](mailto:efi.papatheocharous@ri.se)

## Article info

### Keywords:

Extra-functional properties  
Software engineering  
Modelling  
Ontology  
Software quality  
Evaluation

Submitted: 18 Sep. 2025

Revised: 11 Nov. 2025

Accepted: 19 Nov. 2025

Available online: 26 Nov. 2025

## Abstract

**Context:** The state-of-the-art and practice on software quality is growing constantly and presents several challenges. The ever-growing body of knowledge on the topic, obfuscates the situation further, the lack of explicit structure makes it difficult to identify which properties exist today and how they can be evaluated, two critical aspects to increase software quality.

**Objective:** A step to disambiguate software properties descriptions is made via a Property Model Ontology (PMO) and steps towards the evaluation of the structure are carried out together with experts. The objectives of this paper are: 1) present in detail the PMO, 2) describe the research process used to develop and evaluate the PMO, and, 3) exemplify the usage of the PMO through real instantiations obtained from practitioners and researchers.

**Method:** Expert interviews and qualitative research methods are used to evaluate the PMO and develop a proof of concept.

**Results:** The PMO consists of concepts describing extra-functional properties (EFPs) and their evaluation methods, i.e., how to measure the properties. The PMO is instantiated in a modelling environment through a metamodel and an online web-content management system.

**Conclusions:** Consensus on the definition and structure of EFPs is achieved and a common understanding on how they can be reused in practice.

## 1. Introduction

As defined by IEEE [1], software quality is the degree to which software possesses a desired combination of properties such as reliability, interoperability, performance, maintainability, and usability. In other words, software quality is embodied through software properties. Regardless, the exact list of properties to be prioritised, how they can be defined can be sometimes left open for interpretation. Software properties are commonly and indifferently referred to as extra-functional properties (EFPs), non-functional properties (NFP), quality properties, quality of service (QoS), or simply metrics.

To mitigate fundamental issues with software quality and particularly challenges around the definitions of properties, it is necessary to carry out extensive investigation, clarity and availability activities of subject domain knowledge and this can be performed through

the use of ontologies. Ontologies and their applicability details can enable measurement, comparison and evaluation of a set of critical quality requirements and help reach to improved architectural decisions. Towards this, one needs to specify a systematic (in terms of consistent), understandable and useful way to define “*how to describe*” and “*how to measure*” EFPs as a foundation.

Recent works [2] pinpoint the lack of empirical validations of quality properties and a gap between solutions proposed in academic settings (i.e., the NFR method [3], KAOS [4], QUPER [5], and i\* [6, 7]) and the needs of practitioners [8]. In this paper we bridge this gap by following a dual approach, specifying and validating an ontology, namely the Property Model Ontology (PMO), taking the perspective of academic and industrial validation. The PMO is an ontology which aims to represent the subject domain with as much truthfulness, clarity and expressivity as possible and results to increased expressivity of knowledge, common understanding, clarity of negotiations and problem-solving possibilities for practitioners. An evaluation step is detailed in this work, which includes semantic and pragmatic quality assessments, crucial for structuring knowledge on software quality properties and removing obstacles that obfuscate the state-of-the-art.

An initial version of the PMO for the automotive domain has been described in [9] without however making concrete steps towards its evaluation with experts and collecting feedback from industry. The main contribution of this work is detailing the methodology for developing the ontology, as well as, presenting the evaluation steps carried out through real practical examples. This is a significant advancement from the initial version described in previous work, which did not include concrete steps towards evaluation with experts and collecting feedback from industry.

Since its initial articulation (2016), the PMO has seen limited traction beyond our own pilots. We acknowledge this explicitly and attribute it to (i) the upfront cost for practitioners to model properties formally when informal templates suffice in day-to-day work, (ii) fragmentation of terminology across sub-communities (quality models, measurement, safety/security) which reduces the perceived need for yet another schema, (iii) a lack of tight integration with commonly used toolchains (requirements/issue trackers, architecture decision records, modelling tools), and (iv) scarce, end-to-end exemplars that demonstrate immediate value. In this paper, we report the steps we have taken to address these barriers – publicly releasing PROMOpedia, providing ready-to-use templates, instantiating PMO in a metamodeling environment, and evaluating it on two concrete domains (automotive and security) – and outline further actions to grow adoption (see Sections 5 and 6).

The PMO is a core ontology targeted for the software and systems engineering domain and consists of a structured set of properties to describe software component or system properties, as well as, their evaluation methods, and it has been evaluated through two evaluation rounds, involving in total 10 industrial practitioners and 4 academic researchers. The motivation to develop the ontology is to make a step towards achieving a consensus on the definition and structure of EFPs and share a common understanding on how they can be used. This comprehensive dual evaluation including both semantic and pragmatic quality assessment of the PMO is crucial for structuring knowledge on software quality properties and removing obstacles that obfuscate the state-of-the-art. Such thorough validation is often missing in existing works, making this work a valuable contribution to the field. Another important contribution this work addresses is the issue of scattered knowledge across various communities, domains, and forums. By identifying commonalities and differences, it provides a common ground for fostering discussions and integrating isolated islands of knowledge.

Moreover, to enable software and systems quality assessment the following steps need to be carried out:

1. Establish a prioritised list of EFPs that provide guarantees to the targeted system-level properties;
2. Identify the methods that can be used to evaluate the EFPs;
3. Select to apply one (or several) of the identified methods to evaluate EFPs;
4. Based on the obtained results, identify solutions to achieve the system-level properties aimed for.

In this work, we contribute on the following three aspects: 1) present the evaluated details of the PMO, 2) describe the research process carried out to develop and evaluate the PMO, and, 3) exemplify the usage of the PMO through practical instantiations obtained from industry and academia. The results of these real-world instantiations are presented too.

The remainder of the paper is structured as follows: Section 2 presents the background and motivation based on related work. Section 3 describes the method for constructing the ontology. Section 4 describes the current version of the ontology which is also evaluated in Section 5 by experts and industrial practitioners. The meta-model of the PMO is also described in Section 5. Section 6 discusses the results and lessons learned, limitations and implications for researchers and practitioners. Section 7 concludes the paper and describes future work.

## 2. Background and motivation

### 2.1. Software quality, properties, property models and quality models

A plethora of research on extra-functional properties (EFPs) exists. The body of knowledge is however scattered across many communities, domains and forums, encompassing areas such as testing, requirements engineering, maintainability, decision-making, object-oriented programming, component-based software engineering, model-driven engineering, web-services, software architecture, and many more. This creates many isolated islands of knowledge with limited opportunities for cross-fertilization. Consequently, there is currently no commonly acknowledged definition of what EFPs precisely are. One simple definition can be given by contrasting extra-functional properties to software functionality. Software functionality defines “*what*” a system does, an EFP describes instead “*how-well*” a system performs [10].

The lack of general definitions coupled with the absence of a comprehensive property list and the scattered research results on the topic have led to the creation of many taxonomies and classifications. Properties are generally described in those classifications through a hierarchy of sub-characteristics, which can be further divided into sub-attributes which are verifiable and measurable. For example, reliability in ISO/IEC 9126 [11], is defined as “*a set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time*”, i.e., reliability consists of maturity, fault tolerance, recoverability and reliability compliance.

Some of these classifications are referred to as quality models. As defined by Albin in [12], a quality model is a taxonomy of quality attributes and their relationships. It serves as a framework for detailed system specification, design evaluation, and system testing. Examples of these models include McCall’s [13], Boehm’s [14], the ISO/IEC

9126 [11] and its successor the ISO/IEC 25010 SQUARE quality model [15]. Several other proposals for classifying EFPs have been developed: Laprie’s dependability classification [16], Saraiva’s catalog of maintainability metrics [17] or Crnkovic’s classification of property’s composability [18] are few examples in a long list. An overview of a timeline of existing classifications of software properties can be found in [19]. In [8] a decline in the popularity, relevance and importance of quality models after 2010s has been observed. The authors in [20] try to bridge the gap between abstract quality models/standards and practical tool implementations without addressing the inconsistencies, as pointed out in several studies such as [21–23]. The main problem with these taxonomies, classifications and quality models is that they are typically non-orthogonal, inconsistent among each other or even among themselves, even after decades of research, and lack a common ground as recent systematic literature reviews state [24]. Deissenboeck et al. [23] list several issues as follows:

1. *Ambiguous*. Software quality models generally do not conform to an explicit metamodel, that is, they lack precise semantics, and the interpretations of the concepts and their relationships are left open to the readers. Implied corollaries is that they have:
  - a) *Overlapping concepts*. There are implicit cross-cutting relations between the properties in the quality models which are not captured nor explained. For example, in ISO 25010, security is orthogonal to reliability. However, an unreliable system might negatively affect security, such as when the system is under a Denial-of-Service (DoS) attack.
  - b) *No guidelines*. They do not provide satisfying user’s guidelines on how to use the quality models or how to tailor them for use in a given context.
2. *Limited*. They only support a given view on software quality such as quality in use vs. quality requirements, or system property vs. implementation property.
3. *Arbitrary*. They lack clearly defined decomposition criteria that determine how complex quality concepts ought to be decomposed and the selections of the core properties often seem arbitrary. For example, in ISO 25010, it is debatable that availability should be a sub-characteristic of reliability.
4. *Limited usability*. The properties are too abstract to be straightforwardly used in the verification and validation of a concrete software system (e.g., maintainability [25], usability) and therefore their usability is limited.
5. *Improper metrics*. The supporting metrics often have:
  - a) Unclear motivation and validation [26];
  - b) Ignore context factors which might influence the validity of a property, i.e., the applicability in a given context [27];
  - c) Fail to disclose the impact of the supporting metrics on quality [28].

Glinz [22] identifies similar problems in the classification of non-functional requirements, namely the usage of divergent concepts in the sub-classifications and ambiguity in the types of requirements being classified.

This leads to loose and ambiguous terminology usage [9, 29, 30], uncontrolled and subjective measurement activities and, more notably, to the following issues which obfuscate the state-of-the-art and practice even further:

1. *Synonymous EFPs*. One property can be found under many different denominations, which are in fact completely equivalent (e.g., “*Class Coupling*”, “*Coupling between Objects*” [31]).
2. *Polysemous EFPs*. Two properties can have the same name but refer to two completely different things. The difference can be in semantics, value representation, usage, etc.

For example, the “*DC metrics*” in maintainability can refer either to the “*Degree of Cohesion*” or “*Descendants Count*” [31].

3. *Paronymous EFPs*. Two properties can have the same etymological root, but different meanings (e.g., “composability” and “compositionality”, which both derive from “composition”, but imply different composition qualities of a component assembly).
4. *Incomplete and inconsistent specifications for EFPs evaluation*. The methods used to evaluate the value of a property may not clearly specify what they measure and how. For example, ambiguous ways to measure software size exist, using the physical Source Lines of Code (SLOC) or the logical SLOC. Even though the definitions of the latter vary, SLOC are stated without a clear definition, and depending on the specific use they might include comments and bracket lines, or not [32].

In summary, the topic of classifying software properties is, to a large extent, subject to much controversy and on-going discussions. This makes selecting software components based on properties and evaluating the selections especially difficult [33]. In order to mitigate the challenges around the definition of EFPs and increase clarity in structure and measurement, the use of ontologies is proposed. To the best of our knowledge, no other effort on developing an ontology exists for EFPs that targets together with systematic and formal definitions as well as practical evaluation through industrial practitioners and academics. Our work proposes the PMO as a useful way to describe and reuse software properties and their evaluation methods, and also carries out an evaluation together with experts.

## 2.2. Ontologies in software engineering

In software engineering, the study of human-created knowledge, also known as epistemology, is performed through the use of ontologies. An ontology is a formal, often taxonomic, organization of concepts [34]. A recent literature study identifies an increased use of ontologies in areas of computer science, science and technology and engineering [35], aiming to improve software quality management. Extending ontologies, the article discusses how AI and machine learning can further help create ontologies that update and adapt in real-time in the future. Other recent examples of ontologies for requirements engineering [36] highlight the persistence of ambiguity in natural language for requirements definition.

In the literature and to cope with different purpose and focus, different types of ontologies exist. Their differences have been summarized in several classifications [37–39] although one simple classification differentiates between foundational/high-level/reference ontology and domain ontology, i.e., between generic and specific ontologies respectively.

*Foundational ontologies* are developed with the aim of “*representing the subject domain with truthfulness, clarity and expressivity, regardless of computational requirements*” [40]. The main goal is to help modellers externalize their knowledge about the domain, to make their ontological commitments explicit, to support meaningful negotiations, and to improve the tasks of domain communication, learning and problem solving as best as possible [41]. Simply said, it is a system of categories independent of implementation language or technical constraints [42], and the relation between these categories. Examples of foundational ontologies in software engineering include the Unified Foundational Ontology (UFO) [43], the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [44] and the software measurement ontology [45]. OntoGLOSE, an ontology of the IEEE Computer Science glossary for software engineering, is another example [46].

In contrast, a *domain ontology* is used as a concrete engineering artifact designed for a specific purpose, and represented in a specific language. As identified in [34], this type of

ontology is often inappropriately referred to just an “ontology”. Nowadays, most of the research work dealing with ontologies in software engineering actually belong to domain ontology. This leads to confusions and ambiguity. Examples of domain ontology include the ontology to document a quality scheme specification of a software product [47] and the software maintenance ontologies by Anquetil and de Oliveira [48]. Many more examples can be found in domain ontology repositories such as the Protege Ontology Library<sup>1</sup>.

Ontologies can also be classified according to their level of expressiveness [49]. Similarly to expressive power in programming languages, the level of expressiveness of an ontology refers to its ability to describe the concepts. More expressive ontologies allow specifying all the ontological concepts completely and unambiguously. This is especially useful to highlight the differences between often misunderstood and misused ontology-related terminology. At one side of the spectrum are the informally<sup>2</sup> defined ontologies such as vocabularies, glossaries and thesauruses. A *vocabulary* is a finite list of terms whereas a *glossary* adds definitions in natural language to the terms. *Thesauruses* are glossaries semantic additions such as related terms (e.g., synonyms, antonyms). *Folksonomies* are informal hierarchies created by tagging content. At the other end of the spectrum of expressiveness are the more formally defined ontologies. It includes formal hierarchies, frames, ontologies with value constraints and ontologies with logical constraints. *Formal hierarchies* relate to the concepts of generalisation, inheritance and instantiation in software modelling, by defining an “is-a” relationship between two terms. *Frames* are formal hierarchies in which concepts also include property information. In *ontologies with value constraints*, value restrictions are placed on the properties. Finally, *ontologies with logical constraints* are the most expressive type of ontologies by enabling for example to specify first order logic constraints between terms.

Due to their nature, ontologies provide effective solutions to: i) clarify the structure of the knowledge, ii) reduce conceptual and terminological ambiguity, and iii) facilitate knowledge sharing and communication. As a result, they can play a key role to mitigate current fundamental problems with ambiguity in the state-of-the-art EFP descriptions in software quality and move a step forward in automated intelligent requirement creation and validation.

### 3. Architecting the ontology

Our research methodology followed the four basic steps for ontology construction described in [50], augmented with an additional step for the evaluation. The evaluation step is detailed in Section 5. Our overall methodology process is illustrated in steps in Figure 1.

Step 1 was conducted first, iteratively followed by the execution of step 2, and steps 3 and 4 in parallel. For each iteration of steps 2–3–4, step 5 was then performed. Details on each step is provided below.

#### 3.1. Step 1: Define ontology domain and scope (purpose)

The scope/purpose of the ontology was defined by answering the question “*What is the ontology going to be used for?*”. Initially, we defined the domain and purpose of the ontology to be relevant for targeting to facilitate objective and informed trade-off decisions for

---

<sup>1</sup>[https://protegewiki.stanford.edu/wiki/Protege\\_Ontology\\_Library#OWL\\_ontologies](https://protegewiki.stanford.edu/wiki/Protege_Ontology_Library#OWL_ontologies)

<sup>2</sup>Here formal refers to the ability to be interpreted by a machine.

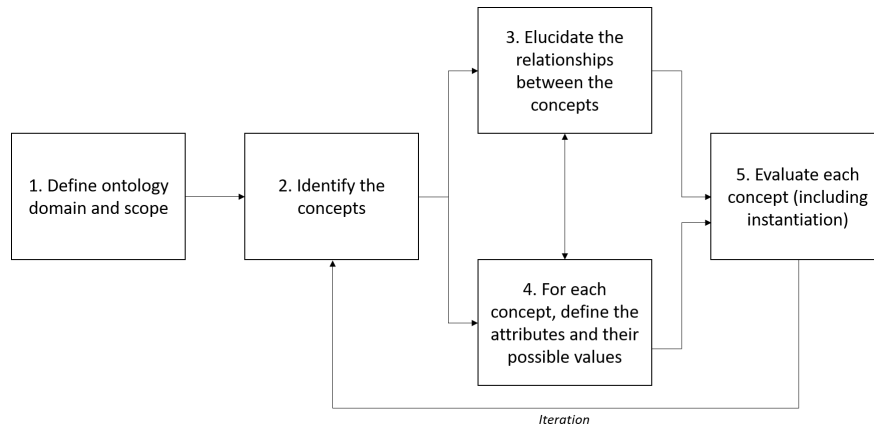


Figure 1. Process for building and evaluating the ontology

software assets based on software quality. The reason for developing an ontology is to explicitly defining the key characteristics of properties and evaluation methods through well-defined taxonomies. An important aspect for the ontology is thus to capture what practitioners need to know for trading off properties to select the most suitable assets and make more informed decisions for software-intensive systems. Answering this, requires identifying of the basic concepts to describe properties and also their evaluation methods (to make possible the trade-off, comparison, estimation, calculation and measurement), which is the next step. However, these aspects are not limited to software-intensive systems. They are also relevant for any type of software systems developed according to software engineering's principles. Therefore, we define our ontology domain and scope (purpose) more generally to any area of software engineering where evaluation and trade-off of software quality is necessary.

### 3.2. Step 2: Identify the concepts

Identifying the concepts naturally entailed ways to describe different types of properties. Given the previous step, the notions included into our ontology were made clear: the concepts we selected needed to support trade-off and evaluation of properties. We have thus identified that “property model” “extra functional properties” and “evaluation method” are the core concepts that should be part of our ontology. The concepts were identified during a workshop held between the authors where after discussion consensus was created. The concepts are tightly related to the purpose of the ontology, i.e., to support trade-off and evaluation of properties, as mentioned above. The reasoning behind selecting these concepts is that they can constitute the basis for collecting definitions and measurement methods within the software and systems engineering domain, that can increase the usability of EFPs and evaluation methods in practice. In addition, as explained in Section 2.1 there is currently no commonly acknowledged definition of what EFPs precisely are and how can they be evaluated. Next step, was clarifying the relationships between the concepts (Step 3).

### 3.3. Step 3: Elucidate the relationships between the concepts

To elucidate the relationships between the concepts of the ontology, a list of questions were developed: “Which extra functional properties should I consider when choosing an

*architectural alternative or component?*”, “*How is suitability of a component evaluated?*”, “*Which critical extra functional properties appear in specific industrial domains, such as the automotive (as an example)?*”, “*How are extra functional properties formalised in scientific publications?*”, “*How is the concept of property models formalised in scientific publications?*”, “*Which aspects of an evaluation method are important that affect its selection for assessing properties?*” and “*How many evaluation methods are preferred to use to sufficiently evaluate extra functional properties?*”.

The questions were developed during the workshop held between the authors where after discussion there was agreement on how the PMO could be used by researchers and practitioners. The questions were used to formulate the relationship between the core concepts and to see whether the ontology contains enough information to answer such type of questions. The list of questions was not exhaustive, in any way, but was used to clarify the required level of details or representation of the concepts and their relationships.

In the related work (Section 2) we mention works that have tried to define the concepts. Additionally, in Section 5 we include a literature survey carried out in the automotive domain to select important properties for the domain and instantiate our PMO.

### 3.4. Step 4: For each concept, define the attributes and their possible values

This step involves establishing for each of the concepts their attributes (what pieces of information are important to describe them) and their possible values. Important questions that we used to carry out this were: “*Which attributes should be used to formalise property models?*”, “*Which attributes should be used to define EFPs?*”, “*Which attributes should be used to describe evaluation methods?*”, “*Are there any existing definitions that can be reused?*”, “*What characteristics should the attributes have?*” and “*What is important to include in these attributes to make them usable?*”. Answering these questions, led us to initial attribute definitions which were then evaluated with specific values.

For the concept of “property” our starting point was the definition proposed in [10]. We realised that since both works served the same purpose, i.e., providing an unambiguous definition and precise semantics for EFPs, we relied on the expert knowledge from that work of one of the authors, to enable our extensions to it. We worked initially as follows: we adapted the definition of the concepts of “attributables” (i.e., the entities of a component model a property refers to) and we disregarded the concept of “support mechanism” (i.e., the set of underlying mechanisms used to manipulate EFPs in a consistent way) for the development of our ontology.

Then, we started by defining the attributes of the concept of “evaluation method”, such as *MethodID*, *Name*, *Output*, and *Unit*. We then instantiated the evaluation methods based on a few evaluation method descriptions that we were familiar with and we also reviewed each other’s interpretation of the descriptions. Details on the evaluation step are in the next subsection (step 5). Based on this evaluation step, we carried out improvements and then we followed an iterative process, where we executed again steps 2, 3, and 4, until no more improvements were identified.

In these iterations we were able to complement the definitions of the concepts with additional attributes gradually (e.g., *Applicability*, *Formula*, etc.) and carry out refinements of the existing definitions. In the end, we created a generic textual template that was used as basis for describing property models in a systematic way.

As previously mentioned, this step was carried out in parallel with the step above, i.e., while carrying out this step, we evaluated the result of our previous step, so as to makes

sure that the core concepts defined were still relevant and sufficient to build upon them our ontology. By enumerating the important attributes for defining our core concepts we were able to expand top-down and then review bottom-up the PMO. The details of our evaluation are provided in Section 5.

Due to the current unstructured state of the body of knowledge in software quality and software engineering, we decided to not provide list of possible values for each of the identified attributes. Defining values is not mandatory for ontologies as it requires having a clear, well-structured and explicit domain knowledge. This is not the case with software quality and software engineering, where many discussions are on-going towards the definitions of core concepts (e.g., software components [21], software assurance [51], software security [52]). As a result, we decide to let the list of values open-ended and with unrestricted format, which facilitate recording currently available information, without compromising the ontology. This restrict the expressive power of the ontology but foster its use.

### **3.5. Step 5: Evaluate each concept (including instantiation)**

The evaluation consisted of two sub-steps and after each of them we carried out amendments to the ontology. These were suggested and then rated (voted individually by the researchers involved in the process) and disagreements were resolved after discussions between the researchers. The two sub-steps performed for evaluation are:

1. Perform a literature survey to find relevant to the industry concepts and attributes (details are found in [9]);
2. Perform an evaluation with academics (collect feedback and instantiations from researchers) and practitioners (collect feedback and instantiations from industrial practitioners).

The details of the steps and the results of the evaluation are presented in Section 5.

## **4. The property model ontology (PMO)**

In this section, we provide the formal definition for our Property Model Ontology (PMO). It starts by defining the concept of property model in Section 4.1, and continues by formalising its key sub-concepts, i.e. EFPs in Section 4.2 and evaluation methods in Section 4.3. The version presented in this section already incorporates the changes made based on the feedback gathered during the evaluations.

### **4.1. Property model**

In this work, we consider a property model as an EFP and the non-empty set of methods that can be used to assess the property. A property model should be described through well-specified characteristics that would allow:

1. comparing different property models;
2. describing an EFP unambiguously;
3. ensuring that an evaluation method is associated with the appropriate EFP definition;
4. comparing different evaluation methods based on facts; and
5. identifying the most suitable methods to evaluate an EFP in a given context (e.g., quality of inputs, applicability in the project, effort).

Thus, we formally specify a property model as a pair consisting of an EFP and a set of evaluation methods so that:  $PropertyModel = \langle PropertyType, Method^+ \rangle$  where,

- $PropertyType$  is the specification of the EFP used in the property model;
- $Method^+$  is a non-empty set of evaluation methods that can be used to concretely assess (estimate, calculate, measure, etc.) the EFP.

In particular, this implies that different evaluation methods can be used for the evaluation of a given EFP. This allows using several competing methods to evaluate an EFP and compare their outcomes.

#### 4.2. Extra-functional property (EFP)

*Properties* are the expression of “how-well” a software entity performs, i.e., its inherent quality. Concretely, a property is a, qualitatively or quantitatively, quantifiable characteristic specified through several elements. Examples of properties that we aim to support in this work include both comprehensive ones, such as dependability, performance, cost and resource consumption, and more specific ones, such as worst-case execution time, development effort, class coupling, etc.

We, thus, defined a property *EFP* by a tuple  $PropertyType$  so that:

$$PropertyType = \langle PropertyID, Name, Output, Unit, Description \rangle$$

where,

- $PropertyID$  is a unique identifier for the property;
- $Name$  is a human-readable name of the property (examples: development effort, reliability, performance, etc.);
- $Output$  is the data type of the property (examples: integer, string);
- $Unit$  is the standard unit of measure or data type for the output of the property (examples: person-month, second);
- $Description$  describes the property in natural language. It must supply enough information to clarify the meaning of the property and its intended usage. Preferably supported by scientific references (Reference(s)).

#### 4.3. Evaluation method

*Evaluation methods* are any approach used to give a value to a given property, be they estimation methods, measurements, mathematical calculations, prediction methods, model-checking, etc.

An evaluation method is defined by a tuple  $Method$  as follows:

$$Method = \langle MethodID, Name, Output, Unit, Applicability^*, Formula, Input^*, Driver^*, Description, Implementation^* \rangle$$

where,

- $MethodID$  is a unique identifier for the evaluation method;
- $Name$  is the name of the evaluation method used for the property (examples: Basic COCOMO 1);

- *Output* is the data type of the output of the evaluation method (examples: number, string);
- *Unit* is the unit of the result or output of the evaluation method (examples: Person-month, second);
- *Applicability\** specifies the elements for which the evaluation method is applicable (example: familiar projects, ambitious projects, tightly constrained/complex projects).
- *Formula* defines how the property is calculated by the evaluation method. It can include mathematical formulae, models, etc.;
- *Input\** is the set of inputs required by the evaluation method;
- *Driver\** is the set of contextual or environmental factors affecting the output of the evaluation method that are not directly involved in the calculation (using the formula). It can include project characteristics, calibration factors, etc.;
- *Description* is the description of the evaluation method including the information necessary to understand and use the evaluation method;
- *Implementation\** refers to available implementation, such as existing tools implementing the evaluation method.

In order to provide useful information to decision makers, the *Description* element is actually a tuple specified as follows:

$$Description = \langle Assumption^*, Advantage^*, Disadvantage^*, Source^* \rangle$$

where,

- *Assumption\** is the list of assumptions the evaluation method is based on;
- *Advantage\** is the advantages of the evaluation method;
- *Disadvantage\** is the disadvantages of the evaluation method;
- *Source\** is the list of references where the description, assumptions, disadvantages and advantages are based on.

#### 4.4. The PMO Metamodel

According to Henderson-Sellers [34], foundational ontologies such as the PMO are equivalent in nature to the metamodel of a modelling language. Therefore, we also specify the PMO in terms of a metamodel to broaden its accessibility and extend its usage possibilities. This is a fairly common practice; previous works have introduced ontology-based metamodels for, e.g., software patterns [53] and agent-based simulations [54].

A metamodel is the set of modelling concepts, the relations among them and the rules by which the metamodel can be instantiated to achieve models conforming to it. Metamodels are at the core of any modelling language, being general-purpose or domain-specific. In this specific case, we define a metamodel that represents the basis for a domain-specific modelling language (DSML) for the description of property models, as this fits the nature of foundational ontologies.

Figure 2 shows the PMO metamodel specified in the Eclipse Modeling Framework (EMF<sup>3</sup>). The concepts specified in the formal definition are directly realised through corresponding entity classes, namely *Property*, *Method*, *Description*, *Input*, *Driver*, *Assumption*, *Advantage*, *Disadvantage*, *Implementation*, *Formula*, *Applicability*, and *Source*.

<sup>3</sup><https://www.eclipse.org/modeling/emf/>

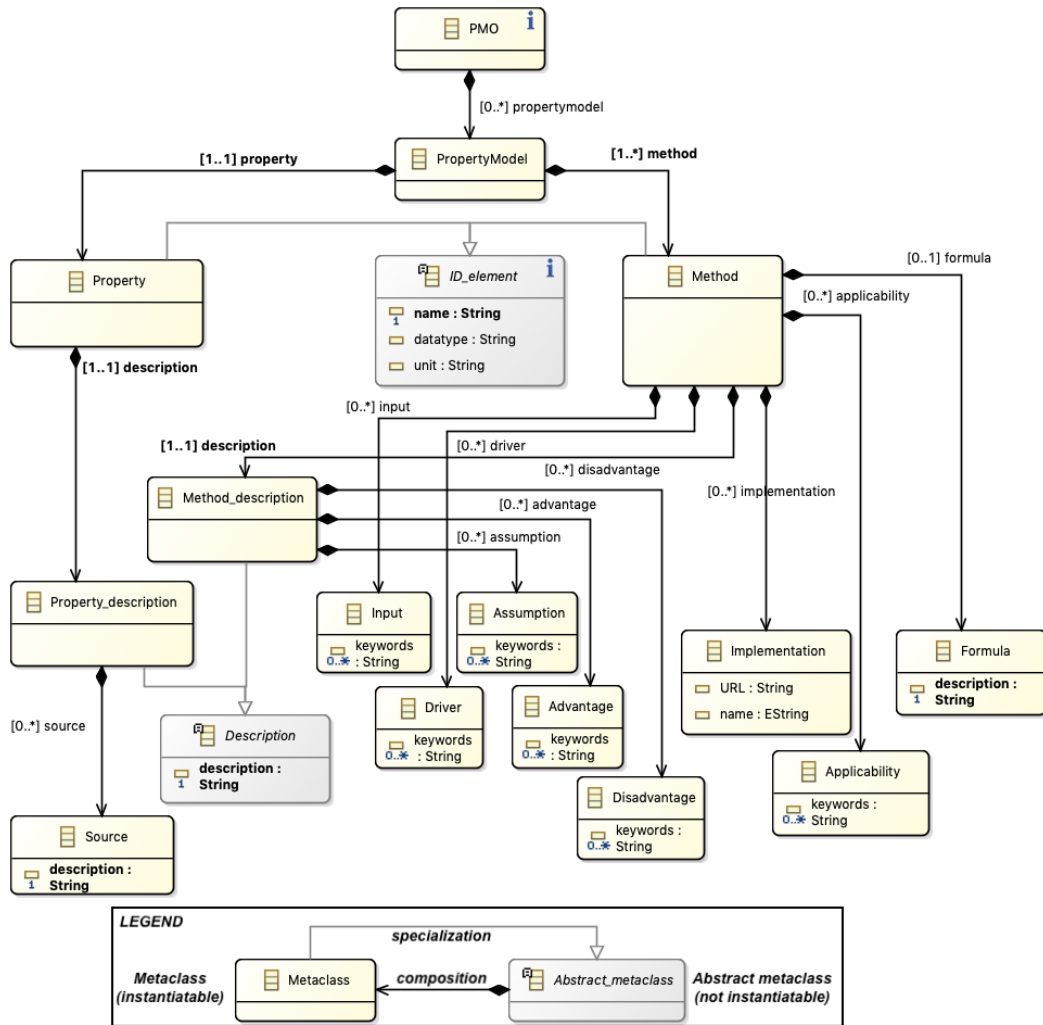


Figure 2. A metamodel in EMF for the PMO

Targeting instantiation, a metamodel offers the opportunity to refine the specification of the concepts through attributes and inheritance to be understandable by a computer<sup>4</sup>. As a result, *Property* and *Method* inherit attributes from the abstract class *ID\_element*: *Name*, *Datatype* and *Unit*. Ideally, *Datatype* and *Unit* should be attributes of *PropertyModel*. However, given that (evaluation) *Method* does not necessarily comply with the output of the property we wanted to be able to record these possible divergences. *Name* is the unique identifier for properties and methods. To completely conform to the formal definition of the PMO, *Name* should have been split into 'ID' and 'Name'. However, the purpose of distinguishing the two is to allow two properties to have the same name, which is not regarded as a good modelling practice.

*Property* and *Method* contain a specialization of *Description*, namely *Property\_description* and *Method\_description*. Other entity classes, more specifically *Input*, *Driver*, *Assumption*, *Advantage*, *Disadvantage*, contains a list of keywords (in string format) for populating them. *Implementation* consists of a *Name* and a *URL*, a string attribute to

<sup>4</sup>Note that, although other formalization notations could be used, such RDF and OWL, we chose to go for an EMF metamodel since it naturally fits in our development tools ecosystem.

encode a url to easily locate corresponding tools for the related evaluation method. *Formula* is currently simply represented by a string attribute where to write a non-empty set of formulae, an algorithm or a software artifact related to the method. We envision a more complex datatype for representing formulae and algorithms. Therefore, we might need to further investigate how to better model this information.

Although the PMO itself in its formal definition could be seen as DSML, describing it in terms of a metamodel within the most commonly used open-source modelling platform for DSMLs definition, i.e., the Eclipse Modeling Framework (EMF), discloses the opportunity to leverage all related model-driven technologies for the manipulation of PMO models. Through model transformations, PMO models can be transformed into input formats for specific analysis tools, be integrated into multi-domain process models, rendered through a wide variety of different notations (e.g., graphical, textual, tabular, web-based), and in general made as integral part of model-driven development processes based on EMF.

The conformity of the metamodel with the formal definition of the PMO was evaluated by instantiating the property models from [55]. It was possible to encode all available data in the instantiation<sup>5</sup>

The purpose of the keyword is to provide a word summary of the descriptions in natural language contained in the element.

#### **Practical examples: applying the ontology concepts in real systems.**

To make the ontology concrete, we illustrate how its core concepts (*Property*, *Evaluation Method* and their attributes) are used in typical engineering scenarios. Each example maps a real decision to PMO fields; the methods correspond to those summarised in Table 4.

##### **Example 1 – Automotive braking-by-wire latency budget (Performance)**

*Property.Name*: End-to-end response time *Property.Output*: number

*Property.Unit*: ms

*Method.Name*: End-to-end response-time analysis over CAN/fixed-priority tasks

*Method.Formula*: combine task/message worst-case terms (e.g.,

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j) \text{ and CAN arbitration delays; see the schedulability}$$

and WCET methods in Table 4.

*Method.Input*: task periods and priorities, WCETs (from IPET/measurement), CAN message sizes and IDs, bus speed.

*Method.Driver*: bus load, scheduling policy (preemption), jitter bounds.

*Method.Assumption*: fixed-priority scheduling; bounded blocking; stable configuration.

*Decision use*: accept if  $R_{\text{brake-chain}} \leq 20$  ms under 90% bus load; otherwise revise priorities or split messages.

##### **Example 2 – OTA update reliability qualification (Reliability)**

*Property.Name*: Failure intensity during system test *Property.Output*: number

*Property.Unit*: failures/hour

*Method.Name*: Musa-Okumoto reliability growth model

*Method.Formula*:  $m(t) = \frac{1}{\theta} \ln(1 + \lambda_0 \theta t)$ ,  $\lambda(t) = \frac{\lambda_0}{1 + \lambda_0 \theta t}$ .

<sup>5</sup>For the interested reader, the PMO metamodel and a sample model can respectively be found at [http://www.mrtc.mdh.se/promopedia/sites/default/files/PMO\\_metamodel.zip](http://www.mrtc.mdh.se/promopedia/sites/default/files/PMO_metamodel.zip) and [http://www.mrtc.mdh.se/promopedia/sites/default/files/PMO\\_model\\_instances.zip](http://www.mrtc.mdh.se/promopedia/sites/default/files/PMO_model_instances.zip).

*Method.Input:* execution time on operational profile, failure occurrence times.  
*Method.Driver:* realism of operational profile; test environment fidelity.  
*Method.Assumption:* independent defect removal; stable debugging process.  
*Decision use:* proceed to pilot if  $\lambda(t_{\text{end}}) \leq 0.05$  failures/hour with 95% CI; else extend testing focusing on high-usage features.

### **Example 3 – Feature sizing for a platform increment (Development Effort/Cost)**

*Property.Name:* Development Effort *Property.Output:* number  
*Property.Unit:* person-months  
*Method.Name:* Basic COCOMO I (or Post-Architecture COCOMO II)  
*Method.Formula:*  $E = a \cdot \text{KLOC}^b$  (mode-dependent constants); Post-Arch adds effort multipliers.  
*Method.Input:* estimated KLOC (or function points), project mode, cost drivers.  
*Method.Driver:* team capability, tool maturity, required reliability.  
*Method.Assumption:* calibrated coefficients for the organisation; 152h/person-month.  
*Decision use:* if  $E_{\text{in-house}}$  exceeds threshold, evaluate *buy/adapt* alternatives; track variance in a release review.

These use case examples show how practitioners take a concrete goal (e.g., meeting a latency or reliability target), select a *Property* with clear *Output* and *Unit*, choose an *Evaluation Method* with explicit *Inputs*, *Drivers*, and *Assumptions*, and then apply an acceptance criterion. Embedding these artefacts in the PMO enables consistent recording, comparison, and reuse across projects.

## **5. Evaluation of the property model ontology**

We first present the research method used for evaluation of the PMO (Section 5.1), then we describe the threats to validity (Section 5.2), followed by the evaluation results (Section 5.3).

### **5.1. Method**

According to [56], there is no single approach to ontology evaluation and the choice depends on the purpose, the application domain of the ontology, and the aspects to evaluate. Burton-Jones [57] describes qualities of ontologies, as well as proposing measurements for the evaluation of the ontology. The main categories of qualities are:

- *Social quality:* Social quality refers to the degree by which the ontology can be relied on and how often it has been used.
- *Pragmatic quality:* This quality focuses on the comprehensiveness/completeness and relevance of the information for the purpose it should be used. In other words, pragmatic quality focuses on determining whether the provided information is useful. It is also strongly linked with expressiveness (cf. [40]), which indicates that the ontology should have the capability to express comprehensive information.
- *Semantic quality:* Semantic quality focuses on whether the terms used in the ontology are meaningful, consistent and clear. Hence, this maps to clarity as defined by [40].

- *Syntactic quality*: Here the focus is on lawfulness (correct syntax) and richness (the complexity of the syntax).

Given that the ontology is relatively new and has not been widely spread in practice and research, we are not able to evaluate its social quality. Syntactic quality is neither in the focus because it is not feasible to check correctness and richness of an ontology that is only instantiated and not widely used in practice yet. Thus, we focus on pragmatic quality as well as semantic quality. Consequently, the following research question is formulated for the validation of the PMO: *To what degree does the ontology achieve pragmatic and semantic quality?*

Despite its first publication in 2016, uptake outside our own studies has remained limited; accordingly, we consider the PMO’s social quality (use and reliance by a broader community) to be low at present. We therefore emphasise pragmatic and semantic quality in this evaluation and return to adoption barriers and mitigation actions in Section 6.

The qualities were first evaluated from the academic perspective (referred to as academic-driven) to identify potential flaws before involving subjects from industry who provided feedback from the industrial perspective (referred to as expert-driven). The research process is shown in Figure 3.

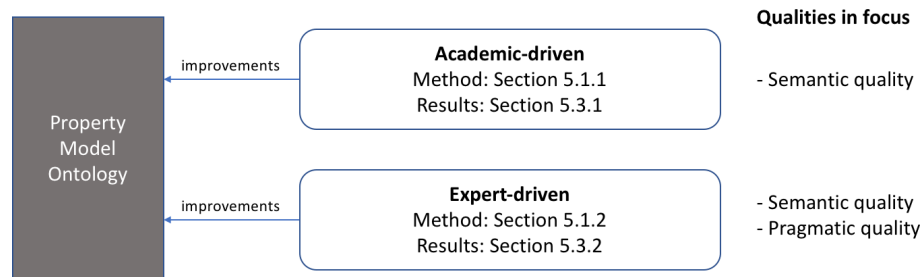


Figure 3. PMO qualities in focus per academic and expert evaluations

#### 5.1.1. Academic-driven evaluation

Prior to evaluating a solution in an industrial context it is recommended to first carry out an evaluation in an academic environment [58]. As the focus during this step is on eliciting the information from the literature, it is not feasible to gather insights from industry about the usefulness of the ontology. Hence, the academic evaluation is focusing on ensuring semantic quality prior to the expert-driven evaluation. We conducted the academic-driven evaluation using the following activities:

1. Identify relevant properties for a selected domain.
2. Identify relevant evaluation methods for the previously identified properties.
3. Classify properties and evaluation methods based on the ontology.
4. Update the ontology based on what has been learned.

*Activity 1. Identify relevant properties for a selected domain.* A first activity towards evaluating the ontology was the identification of relevant properties and evaluation methods for these properties. As the proposed ontology was developed in the context of the research project ORION the initial emphasis was on identifying properties within the automotive domain, which was in the focus of the project. Priority was given to these properties which were most frequently mentioned in the literature for the automotive domain [9].

*Activity 2. Identify relevant evaluation methods for the previously identified properties.* After a set of relevant properties was identified, we gathered three additional papers for the most frequently mentioned properties in the automotive literature and collected descriptions of their evaluation methods. The focus was on identifying real industrial cases, in at least one domain (here automotive), where evaluation methods are identified and discussed. The identification of such cases would enable us to obtain and reuse domain knowledge. Also, as our ontology is a model of the real world, so the concepts defined in it must reflect reality, and thus it is important when we defined an initial version of the ontology, to evaluate it using a real application or problem as suggested in [50]. Therefore, in our evaluation we followed the suggestions of Noy and McGuinness [50], and took into account real cases of considering EFPs in an actual domain, i.e., the automotive domain.

*Activity 3. Instantiate the ontology for different properties and evaluation methods.* For this purpose two iterations were realized to record data:

- Iteration 1: Instantiation with well-known properties and evaluation methods relevant to the automotive domain
- Iteration 2: Instantiation with security properties and their evaluation methods [59].

For this purpose two instruments were created to record data: a text document for iteration 1, where the data could be easily extracted and comments could be added, and a spreadsheet for Iteration 2.

*Activity 4. Update the ontology based on what has been learned.* The comments from Activity 3 were discussed among the researchers and a rejoinder was created to decide how to address the concerns raised.

Table 1 provides a brief overview of the researchers participating in the evaluation.

Table 1. Evaluation participants (researchers)

Id	Position	Research Topics
R#1	Assoc. Prof.	Component-based software engineering, model-based development, embedded systems
R#2	Senior researcher	Cost estimation, software architecture, agile software development
R#3	Prof.	Software processes, software metrics, empirical software engineering
R#4	Senior researcher	Software quality, enterprise architecture, cyber security

The revised ontology then was used as input for the interviews with the practitioners (expert-driven evaluation).

### 5.1.2. Expert-driven evaluation

The evaluation was completed with interviews of practitioners who have spent at least five years working in industry. Their demographic data collected is shown in Table 2. The partial elements filled-in the table are due to inability to collect all the data during the interviews due to technical failure of the system used for the survey. Interview from I#9 was also not carried out and thus is removed from the sample. The interviewees from industry were asked to reply to questions of an online survey<sup>6</sup>, whereas one researcher was observing the process. The researcher was taking notes of any additional information or reflections that interviewees mentioned during the process of entering information.

In the beginning of the online survey, some general information was given about the survey, and the interviewees were asked to provide information of a specific (but of their

<sup>6</sup>The survey elicitation form is available here: <https://goo.gl/forms/6OfIjo4K1mfaypsR2>.

Table 2. Interviewed experts from industry

Id	Position	Domain	Property	Evaluation Method
I#1	Tester/Developer	Telecom	Reliability	[None]
I#2	Product Manager/ Tester/Developer	Telecom	Maintainability	Peer review
I#3	QA	Transport	Safety	Safety standards assessment
I#4	Consultant/Researcher	IT consulting	Performance	Timing analysis
I#5	QA/Developer	Fin-Tech	Timing	Critical path response time
I#6	Researcher	IT consulting	Timing	Response time analysis
I#7	Tester/Researcher	RT systems	-	-
I#8	Researcher	Automotive	Timing	End-to-end timing analysis
I#10	Consultant/Researcher Researcher	IT	Development effort	Regression analysis
I#11	Consultant/ Researcher	IT	Development effort	Function points or objective points

choice) property model, evaluation method related to the chosen property and finally, general feedback. The feedback related to evaluation of the semantic quality of the elements and their evaluation methods, including how informative they are. The feedback also related to evaluation of pragmatic quality (i.e., focusing on usefulness) of the PMO. Pragmatic quality included utility scenarios in practice. Moreover, interviewees were asked to identify unclear and/or missing elements. During the data collection, as well as in the feedback part of the survey, we were able to collect change requests to specific PMO concepts and the survey itself, to help us improve the PMO.

The change requests helped in evaluating both the semantic and pragmatic quality of the ontology. The number of changes requested is an indication for the quality of the ontology. If the number of changes is high then the ontology needs to be further revised to improve its quality, both semantically and pragmatically. On the other hand, if the definition of an element needs to be changed for the element to become clear and meaningful, then the ontology needs to improve on semantic quality.

Overall, we counted the number of changes and classified them into four groups, namely structural, definition, example and reformulation, which are defined as follows:

- *Structural*: The structure of the ontology changes. New elements need to be added or existing elements need to be removed. It also includes changes where one element needs to be split. Structural changes are primarily linked to pragmatic quality as information needs to be added or removed to increase the relevance of the ontology.
- *Definition*: The definition (i.e., description) of an element of the ontology has to be changed for the element to become meaningful. This concerns semantic quality.
- *Example*: Together with the definition of the elements, an example was given. The example used for the elements of the ontology needs to be changed. Examples help to improve the understanding and hence clarity of the ontology (semantic quality).
- *Reformulation*: An element needed to be reformulated, though its meaning (definition) has not changed. Reformulations are mainly focused on increasing clarity (semantic quality).

The above list shows the change types according to their complexity and gravity, from high to low; i.e., a structural change requires to make more significant changes to the ontology structure and probably rework multiple elements, whereas a reformulation can be

more simple, involving cosmetic changes and does neither change structure nor meaning of elements.

## 5.2. Validity threats

The threats of validity were identified and evaluated during the research, based on the proposals by Maxwell [60] and the derived classification of threats by Petersen and Gencel [61], with a particular focus on qualitative research.

### 5.2.1. Descriptive validity

Descriptive validity refers to our ability to describe what is happening accurately. In the context of the study it is important to capture the interviews accurately. This was supported through the online survey where practitioners recorded their answers and also provided their comments. To avoid misunderstandings during the completion of the interview it was conducted face-to-face. Hence, the threat is considered under control.

### 5.2.2. Theoretical validity

Theoretical validity is threatened if we cannot capture what we intend to capture, e.g., due to poorly defined constructs or confounding factors. During the evaluation in the academic context members of the ORION project took the roles of the evaluators. Hence, they are familiar with the ontology and hence lack an external perspective. The evaluation, however, adds confidence prior to using the ontology externally with practitioners. We also did not only rely on the academic evaluation, but complemented it with the perspective of practitioners.

### 5.2.3. Generalizability (internal and external)

Internal generalizability refers to the ability to generalize within a context and external generalizability to other contexts. During the academic evaluation the target was a particular industrial domain, the automotive domain. Hence, a restricted set of properties and evaluation methods was in the focus. Within that domain, a range of methods has been elicited and documented. Hence, the internal generalizability is considered high. However, the external validity of the study is limited. The threat was to some degree reduced by complementing the work with practitioners from different domains (e.g., Telecom, see Table 2).

### 5.2.4. Interpretive validity

Interpretive validity is concerned with drawing valid conclusions from the data. A potential threat are the biases of researchers during the interpretation and analysis of the data. To reduce the threat we utilized observer triangulation during the academic evaluation and the practitioner interviews. In decisions and interpretations all authors were involved. Also, all comments, responses and decisions were recorded in a rejoinder, making the design decisions for updating the ontology traceable.

### 5.3. Results

We first present the results of the evaluation in the academic context, followed by the results of the evaluation with industry practitioners.

#### 5.3.1. Evaluation in the academic environment

We show the results of each individual step of the evaluation in the academic environment. The first two steps (identify relevant properties and evaluation methods) of our research have already been presented in [9], while the systematic analysis of the feedback and subsequent updates to the ontology have been added in this study as a preparation for the industrial evaluation. We shortly repeat the main findings of our previous paper to ensure the traceability of the research steps.

*Activity 1. Identify relevant properties for a selected domain.* We identified nine properties and mentioned the related papers in which the properties have been presented. We also saw that specific properties have been present in most of the research papers. Table 3 summarizes the results of this study, whereas more details can be found in [9]. Numbers in the table highlight the priorities among the EFPs identified as relevant in the referenced literature, whereas a “+” symbol means that the property was identified as important but no explicit ranking was provided.

Table 3. Relevant EFPs for the automotive domain

Properties	[62]	[63] Scania	[63] Volvo	[64]	[65]	[66]	[67]	[68]	[69]	Final Ranking
Cost	1	1	1	+		+	+	+	+	#1
Performance	2	2	3	+	+		+	+		#2
Configurability, Variability	3			+						
Flexibility, Maintainability, Testability	4	6		+						
Power Consumption	5		2						+	
Reliability, Safety		3, 5	4	+	+	+	+	+	+	#3
Evolvability		4								
Security							+			

*Activity 2. Identify relevant evaluation methods for the previously identified properties.* For each property identified in the previous step, several evaluation methods have been extracted (in total 19 evaluation methods) as shown in Table 4. The researchers extracting which models are shown in the same table.

*Activity 3. Classify properties and evaluation methods based on the ontology.* The researchers extracted the property descriptions and corresponding evaluation methods (summarised in Table 4, also detailed in [55]).

To illustrate how the ontology is instantiated Table 5 presents three evaluation methods for development effort, namely the Basic COCOMO I, Estimation by Analogy (EbA) and Expert Estimation. Each model is described with its attributes and corresponding values. For example, the Basic COCOMO I is applicable for in-house development, EbA requires the availability of data from similar projects and expert estimates require experienced practitioners (values). The ontology also describes the method narratively (description) and provides the formula(s) to be used. Knowing the inputs may also provide valuable insights on

Table 4. Researchers, properties and elicited evaluation methods

Id	Property	Evaluation Method
R#1	Performance	Controller Area Network (CAN) schedulability analysis, End-to-end response time (or delay) analysis, Worst-case execution time (WCET) analysis based on Implicit Path Enumeration Technique (IPET), Measurement-based WCET analysis, Probabilistic Hybrid WCET Analysis, Probabilistic Worst-case Response-Time Analysis for CAN, Worst-Case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption
R#2	Cost	Basic COCOMO 1, Intermediate COCOMO 1, Advanced, Detailed COCOMO 1, Expert Estimation
R#3	Reliability	Logistic model, Inflection-S model, Musa-Okumoto, Jelinski-Moranda
R#4	Availability	Steady state availability analysis based on MTTF and MTTR, Component-based availability analysis, Markov state space models

whether we are able to apply the model (e.g., the ability to determine the required inputs). The drivers affect the output of the model beyond the formula and hence are confounding factors. For COCOMO these factors are explicitly defined and classifiable (development mode, application domain, project characteristics), while for expert estimation the drivers are mentioned, but do not have a classification/ measurement scale. Assumptions of using the methods (in particular with regards to statistical approaches) are important to not be violated, as it may result to invalid outcomes. Moreover, the advantages and disadvantages are stated. Finally, the source and implementations of the method are listed.

Based on the evaluation forms the four researchers provided their reflections using the data collection form. The reflections focused on the **semantic quality** of the ontology and improvements were made based on the feedback collected.

Overall, the PMO was instantiated with seven properties (development effort, development time, cost, performance, reliability, availability and security) with more than 40 evaluation methods (e.g., Post-Architecture COCOMO II, CAN schedulability and Worst-Case Execution Time), details of which can be found in [55].

*Activity 4. Update the ontology based on what has been learned.* For each comment raised by the reviewers a response was written and a decision was taken among the authors how to address the issue.

Hence, prior to the evaluation with the practitioners, the internal evaluation did not lead to major changes in the ontology, but to several improvements towards clarity. This also provided confidence in the ontology as a variety of evaluation methods was classified using it by different persons.

### 5.3.2. Result from interviews with practitioners

As mentioned in Section 3 the feedback during the evaluation was considered for the identification of concepts and revising their relationships and their attributes. Table 6 shows the number and type of changes collected from the feedback per each PMO concept. In total 28 change requests were made, a small figure compared to the vast majority of the concepts receiving no change request (82.5%). The change requests were implemented after an agreement process followed by the authors of the paper which included reviewing, discussing, suggesting changes and agreeing on the changes proposed, until no concepts were left to be agreed and all proposed changes were agreed.

**Semantic quality.** The evaluation of the presentation of the PMO in terms of informativeness, i.e., whether it served its purpose it was created for, was positive by the

Table 5. Excerpt of the property model for development effort

Property Name*		Development Effort	
MethodID	Basic_Cocoma_I	Analogy_Estimate	Expert_Estimate
Name	Basic Cocomo I	Estimation by Analogy (EbA)	Expert Estimation
Output	number	number	number
Unit	Person-month	Person-month	Person-month
Applicability	Familiar projects, ambitious projects, tightly constrained/complex projects, in-house development	Anytime, similar data is available	Similar projects, expertise and experience, to any case
Formula	$a(KLOC)^b$ , where $a$ and $b$ are two constants which are set based on the development mode.	N/A	N/A
Input	– KLOC (thousands lines of code)	N/A	– Expert’s value estimate – Expert’s confidence, experience, skill – Expert’s weights – Expert’s background – Available checklist of information related to the project
Driver	– Development mode (organic, semi-detached, embedded) – Application domain – Project characteristics: Size, Innovation, Deadline, Dev. Environment	N/A	– Expert’s experience and knowledge – Gut-feeling
Description	The method computes software development effort as a function of program size expressed in estimated kilo-line of code	The method estimates the property by analogy with similar completed projects in the same application domain	The method uses a person recognized as an expert on the task to estimate the value. Typically, it relies on tacit knowledge.
Assumption	– Configured for 152hours/Person-month.	– Prior knowledge exists (stored or can be accessed).	– Expert is available
Advantage	– Transparency – Good for quick, early and rough estimate – Drivers provide insights to the decision maker on what affect the project	– Clear mathematical reasoning – Provides opportunities for weighing and obtaining local and global measures	– It is one of the dominant strategies when estimating software development effort – Simple to understand – Does not rely on size metrics (e.g. KLOC) to provide the estimate
Disadvantage	– Limited accuracy. – Difficult to estimate KLOC early in the project. – Vulnerable to the quality of model tuning.	– Similar projects exist and were documented. – A priori selection of the most important attributes.	– How the estimation has been performed remains typically undocumented. – Subjective to the expert’s opinion and bias.
Source	[70]	[71]	[72]
Implementation	N/A	N/A	N/A

\*For simplicity the Property details are omitted as they match the *Output*, *Unit* of the Method

Table 6. Change requests for each PMO concept

PMO concepts	Structural	Definition	Example	Reformulation	No change
<i>Property.Name</i>	1	1	1	1	7
<i>Property.Output &amp; Property.Unit</i>	0	0	0	0	9
<i>Property.Description</i>	0	0	0	0	10
<i>Method.Name</i>	1	2	0	0	8
<i>Method.Output</i>	1	4	0	1	4
<i>Method.Unit</i>	0	0	0	0	9
<i>Method.Applicability</i>	0	5	1	0	4
<i>Method.Formula</i>	0	0	0	0	10
<i>Method.Input</i>	0	1	0	2	7
<i>Method.Driver</i>	0	2	0	0	8
<i>Method.Description</i>	0	0	1	1	8
<i>Method.Assumption</i>	0	0	0	1	9
<i>Method.Advantage</i>	0	0	0	0	10
<i>Method.Disadvantage</i>	0	0	0	0	10
<i>Method.Source</i>	0	0	0	1	9
<i>Method.Implementation</i>	0	0	0	0	10
Total	3	15	3	7	132

interviewees (in 8 out of 10 cases). One case (out of the 8) commented that it was not fully informative what the aim of the PMO was or which properties were mostly important (I#11). Regarding understandability of the property and evaluation methods concepts a likert-scale was given to the interviewees from 1–5 (1: strongly agree, 5: strongly disagree). On average (and mean) understandability was ranked at 2 (by 8 interviewees, as values were not available from all questionnaires), indicating partially high understanding.

Another respondent found some of the standards/patterns in the options a little unclear (I#2). Another request was for more elaborate baseline descriptions (i.e., on the aims of the PMO and their evaluation methods) as well as allowing more possible ways for filling-in responses (I#11). The same interviewee (I#11) also commented not liking to have to write so much, something that highlights the need for more elaborate property and evaluation methods descriptions to be made available. I#6 commented on how the questions were posed in the online form and didn't really understand the need for using such elicitation method for collecting data. I#10 commented on the use of the word "property" not reflecting well the term "variable" typically used in estimation, analysis and assessment contexts. However, the respondent was able to overcome this issue through the examples given throughout the questionnaire, commenting that they were very helpful, made the questions clearer and less ambiguous. Overall, in order to improve understandability, interviewees requested for 3 structural, 15 definition, 3 example changes and 7 reformulations of the questions (see Table 6) and these were implemented by the authors.

**Pragmatic quality.** The pragmatic quality of the PMO was evaluated by requesting some envisioned scenarios of use. I#10 would use them for "comparison of evaluation methods or for combining the output of different evaluation methods" to prioritise, for example, needs. I#11 was more specific in the PMO usage as "an analysis of effort estimation methods with around 15 characteristics, for example as the one found in Pfleeger et al. [73].

Along the same lines of thinking, I#1 found the PMO useful if it "can provide guidelines based on the properties, e.g., what mathematical models to use an when" which is a good distinctive value offered by the PMO, as identified by its receivers.

I#4 and I#7 found the PMO useful in several scenarios and especially as a reference or as a reference guide to definitions of properties and their evaluation methods. I#8

found it useful and helpful for their automotive customers. I#2 explains that if PMO “reduced review time and helped out in the automation of the review (i.e., according to pre-defined standards) then it would be really useful.” I#3 said “it sets the scope and gives and understandable frame” and I#5 explained “it can help two individuals with a different understanding [of a property] to reach to a common language. A catalog of properties is something I often looked for my PhD studies.”, whereas I#6 also agreed “it can be very useful especially for finding agreement in case of discordance.”

Interviewees were asked for any information or concepts missing from the PMO. I#8 suggested that the context of the application domain might be important when discussing properties and I#4 the stage of development at which the evaluation method is applied/applicable. These concepts are however already included in the PMO, under *Method.Applicability*. I#1 suggested that we consider just the context of a team and think of ways to include properties related to team factors, such as performance and skills. I#10 suggested measures of accuracy as a concept missing from the PMO which could be a way for comparing the results of different evaluation methods (e.g., mean absolute error, or mean relative error). I#11 considered as nice to have a method to judge or provide input on a pre-made analysis of evaluation methods. It would also be good if expert-based methods were also described. The rest of the interviewees did not have anything more to add to the PMO.

## 6. Discussion

### 6.1. Summary of key findings

The impact of the academic evaluation, mostly targeting **semantic quality** of the ontology was rather low, as it did not lead to major changes to its structure or definition. The subject domain was representative (based on the feedback collected) and the meaningfulness was found high, as very few changes to improve clarity were proposed. The improvements mentioned were about more well-defined elements. Both the classification of the fields as optional or mandatory and the use of short examples, as requested by the researchers, addressed the issues of clarity and consistency. Therefore, the semantic quality of the PMO was found as satisfactory to proceed to the industrial evaluation.

The industrial evaluation resulted to a small number of changes (only 8% vs. 92% of no change requests) showing a high acceptance rate with regards to semantic quality of the PMO by the practitioners. Understandability was ranked on average as high. A few improvements that were requested concerned the level of elaboration for the elements (i.e., related also to consistency), something that is a concern in all crowd-sourced material, although we attempt to address it through the use of examples. Overall, the industrial experts were positive towards our approach to the definition of properties and evaluation methods, which provided us the confidence to proceed and develop PROMOpedia for subsequent evaluation of pragmatic quality.

### 6.2. Reflections on the quality of the ontology

Concerning the **pragmatic quality** of the PMO, the results of the interviews consolidated our idea of the potential usefulness of the PMO for data extraction purposes when carrying out systematic studies. The PMO was in fact perceived by interviewees as a reference guide

to definitions of concepts, a way to properly set scope and frame of specific aspects under study, and in general a way for individuals with different understandings and coming from different domains to reach a common language. The purpose of systematic studies is to present a body of knowledge in an unbiased and objective manner, and the PMO could be a useful tool to report on software-related properties. We evaluated this by employing the PMO to systematically investigate and report how software security-related properties are evaluated [52]. The PMO was used as backbone of the data extraction phase of our systematic study and proved itself to be a very useful tool in this respect, as initially presumed. Moreover, we perceived that the way the PMO is structured helped us in methodically retrieving information in an efficient manner although none of the authors had previous experience with the specific set of properties (related to security) under study.

With respect to improvements in the PMO structure, changes to the structure and to the definitions were carried out based on the feedback collected. These changes materialised improvement in the PMO, however still some issues remain. Ideally, the evaluation methods' unit and output should match the ones of the corresponding properties. However, this is often not the case. Therefore, we opt to record both of them in order to enable identifying possible mismatch in the state-of-the-art and practice. There's an implicit relation of the applicability and drivers used in the PMO with the context of where an evaluation method should or could be used. However, our work focused on implementing and evaluating the ontology and has not investigated in depth this relation.

### 6.3. Reflections on industry adoption

**Adoption to date and barriers.** While the academic and practitioner evaluations indicate that the PMO is understandable and useful as a structuring device, *its social traction has been modest since 2016*. In our experience, four factors have hindered broader uptake: (1) *On-ramp cost*-encoding properties and evaluation methods demands time and ontology literacy with unclear short-term payoff; (2) *Terminology fragmentation*-adjacent communities (e.g., ISO/IEC 25010 quality models, software measurement, safety/security) already use entrenched vocabularies, creating perceived overlap; (3) *Toolchain fit*-lack of out-of-the-box integration with requirements and modelling tools reduces everyday utility; (4) *Evidence gap*-few publicly available, end-to-end exemplars that show PMO accelerating decisions or reducing defects.

**Actions taken to increase adoption.** We have pursued several concrete steps: (i) *Open availability* via PROMOpedia with ready-to-use templates to minimise authoring overhead; (ii) *Executable artefacts* by instantiating the PMO as a metamodel in a modelling environment, easing reuse in projects; (iii) *Worked exemplars* beyond automotive (e.g., security-related properties) to demonstrate cross-domain applicability; (iv) *Process embedding* by trialling PMO templates in internal courses and project reviews. *Ongoing efforts* include packaging a small “PMO-core” subset for quick starts, providing machine-readable exports (e.g., JSON/OWL) and mapping guides to adjacent standards to reduce cognitive friction, and exploring light-weight integrations with requirements/issue trackers to meet practitioners where they work.

### 6.4. Comparison with related work

In relation to the concept of “property model” several scientific works exist which define it, i.e., [70, 74–79]. They point to interesting aspects of the concept but they do not touch

upon our vision and use of the concept of property model as a key factor to evaluate the suitability of a component from an extra-functional property viewpoint. For example, [77] only focuses on functional property models. In [74], the concept refers to a property only viewed as a constraint that can be verified whereas in [75–77] it is used to characterise a system through its perceived properties. In [78], the concept of property model is meant to be a formalisation of a set of properties (resources, mechanisms, policies) exploited at deployment time for the various deployment participants to share the same property semantics. Besides, the examples of property models used in [75] mix properties, evaluation models, metrics for properties.

Our definition of property and methods is deliberately broad and inclusive to encompass the different terminologies related to extra-functional properties and enable capturing their nuances. For example, in the case of worst-case execution time, two properties can be defined using the formal language specified in Section 4: one which focuses on the time as the data to be processed (i.e.,  $\langle$  “WCET”, “Worst-case execution time”, “Real number”, “s” $\rangle$ , [80]) and one on the processor’s clock cycle (i.e.,  $\langle$  “WCET\_2”, “Worst-case execution time”, “Real number”, “clock cycle”, [80] $\rangle$ ). This enables easily associating the right evaluation methods to the right property and limit incompatibility risks. Furthermore, for many properties, many different definitions exist, often highlighting specific details. Again, for worst-case execution time, the following definitions can be found: “*The worst-case execution time (WCET) is the longest time it takes to execute a given program code* [81],” “*The worst-case execution time (WCET) is defined as the longest execution time of a program that could ever be observed when the program is run on its target hardware* [82].” As shown with both examples, i.e., with the differences in data and the differences in definitions, this has actual implication on the evaluation methods and the amount of information necessary to calculate the property and its validity in different context.

Being inclusive can potentially lead to information bloat. However, given that one of our primary goal with this research is to provide a starting point to unravel the state-of-the-art on software properties, listing up all available definitions is an important necessary step. It would allow identifying commonalities, differences and provide a common ground for fostering discussions.

## 7. Conclusion

In this work we presented in detail the Property Model Ontology (PMO) and how it was devised. We described its evaluation together with experts and exemplified its use. The evaluation carried out included semantic and pragmatic quality of the PMO and we conclude that: (a) the steps carried out towards an evaluation together with experts is a difficult activity to carry out, especially for a new ontology, having lack of wide and effective usage by practitioners; however it is a mandatory and important step to structure the knowledge on software quality properties and remove the obstacles that obfuscate the state-of-the-art; (b) by focusing on the assessment of pragmatic and semantic quality of an ontology we need, as next step, to assess social and syntactic quality to obtain an even more generalizable and complete evaluation.

Overall, we have achieved a consensus from the experts, on how to capture EFPs using the concepts, attributes, and relations in the PMO, consensus was reached on the concepts in the PMO and a shared understanding on its possible uses in practice. The definition of the PMO, provided in this paper, offers a framework so that individuals and organizations

may define their own property models and describe the related evaluation methods. They can form the preconditions for assessing quality in the development of software-intensive systems. While doing so, however, it is up to each individual or organization to define in precise detail and values of their own model according also to their scope. For instance, one usage may be to select assets from multiple alternatives based on specific values or range of values for EFPs which are considered acceptable by the organization.

The PMO is an ontology which aims to represent the subject domain with as much truthfulness, clarity and expressivity as possible. We support that it can result to increased expressivity of knowledge, and clarity of negotiations for practitioners. Moreover, based on the classification proposed in [49], the PMO also belongs to the frames category. However, by fostering its use through PROMOpedia, the online encyclopedia of property models for software engineering [83], our long term objective is to be able to identify set of value constraints for the different entities, hence transforming the PMO to an ontology with value constraints. The PMO can also be used for representing a domain body of knowledge, resulting from systematic studies for individuals with different understandings of properties and their evaluation methods to reach to a common language and improved clarity.

Recent rise in Artificial Intelligence (AI) and in particular GenAI (Generative AI), offer significant potential to support and enhance the development and application of application and domain-oriented PMOs. However, the lack of large dedicated datasets from multiple sectors and industries inhibits this and to fully reaping the benefits. Advances in Domain-Specific Languages (DSLs), may assist in automating parts of the PMO instantiation process for specific domains, where the relevant knowledge is available and well-documented. For example, GenAI can assist in generating tailored DSLs that encapsulate the key concepts, relationships, and value constraints defined within a PMO. This approach can facilitate in the creation of expressive, user-friendly languages for practitioners, enabling them to interact with and extending PMO-based models more efficiently. AI/GenAI can also assist in maintaining consistency and clarity between different PMOs instantiations by working with terminology and generating supporting tools, like chatbots, parsers and validators.

As future work, we plan to use the ontology and investigate further its expressivity capabilities, content clarity and quality, as well as identify means to enroll the research, academic and practitioner communities to foster its metamodel and instantiations through for instance, PROMOpedia. Future work will focus on concrete evidence of the PMO's practical utility and adoption, with comprehensive end-to-end case studies. A proposed approach could be employing A/B testing and comparing results between practitioners using PMO and not using it, to measure improvements in decision quality, speed and required investment effort. This approach will allow the generation of empirical data on the benefits and adoption barriers of the PMO in real-world conditions. Furthermore, we will seek to engage the research, academic, and practitioner communities in evolving the PMO metamodel and its instantiations through the PROMOpedia platform. Community involvement will be essential in refining the PMO's usability and ensuring that it meets the practical needs of its users.

## Acknowledgment

Acknowledgements to our colleagues in the ORION project and the participants in the evaluation for the fruitful discussions and contributing in improving the work. We especially

acknowledge the contributions and inspirations from the late Séverine Sentilles throughout this work.

## CRediT authorship contribution statement

E Papatheocharous: Conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing – original draft, writing – review and editing. S. Sentilles: Conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing – original draft, writing – review and editing. K. Petersen: Data curation, formal analysis, investigation, methodology, validation, writing – original draft, writing – review and editing. F. Ciccozzi: Conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing – original draft, writing – review and editing.

## Declaration of competing interest

There are no competing interests.

## Funding

The work is supported by a research grant for the ORION project (reference number 20140218) from The Knowledge Foundation in Sweden.

## References

- [1] N. Schneidewind, *Standard for a software quality metrics methodology*, IEEE Std. 1061-1998, 1998.
- [2] T. Olsson, S. Sentilles, and E. Papatheocharous, “A systematic literature review of empirical research on quality requirements,” *Requirements Engineering*, Vol. 27, No. 2, 2022, pp. 249–271.
- [3] J. Mylopoulos and L. Chung, “Representing and reasoning with non-functional requirements: A retrospective,” *IEEE Transactions on Software Engineering*, 2025.
- [4] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*, Vol. 5. Springer Science and Business Media, 2012.
- [5] B. Regnell, R.B. Svensson, and T. Olsson, “Supporting roadmapping of quality requirements,” *IEEE Software*, Vol. 25, No. 2, 2008, pp. 42–47.
- [6] E. Yu, “Towards modelling and reasoning support for early-phase requirements engineering,” in *International Symposium on Requirements Engineering*, 1997, pp. 226–235.
- [7] X. Franch, J.C.S. do Prado Leite, G. Mussbacher, J. Mylopoulos, and A. Perini, *Social Modeling Using the i\* Framework*. Springer, 2024.
- [8] I.G. Ndukwe, S.A. Licorish, A. Tahir, and S.G. MacDonell, “How have views on software quality differed over time? Research and practice viewpoints,” *Journal of Systems and Software*, Vol. 195, 2023, p. 111524.
- [9] S. Sentilles, E. Papatheocharous, F. Ciccozzi, and K. Petersen, “A property model ontology,” in *Software Engineering and Advanced Applications (SEAA)*. IEEE, 2016, pp. 165–172.
- [10] S. Sentilles, *Managing Extra-Functional Properties in Component-Based Development of Embedded Systems*, Ph.D. dissertation, Mälardalen University, Västerås, Sweden, 2012.

- [11] *Software engineering – Product quality*, ISO/IEC Std. 9126, 2001.
- [12] S.T. Albin, *The art of software architecture: design methods and techniques*, Vol. 9. John Wiley and Sons, 2003.
- [13] J.A. McCall, P.K. Richards, and G.F. Walters, “Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality.” GENERAL ELECTRIC CO SUNNYVALE CALIF, Technical Report ADA049014, 1977. [Online]. <http://stinet.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA049014>
- [14] B.W. Boehm, J.R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in *Proceedings of the 2nd international conference on Software engineering*, ICSE ’76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 592–605. [Online]. <http://dl.acm.org/citation.cfm?id=800253.807736>
- [15] *Systems and Software Engineering. Systems and Software Quality Requirements and Evaluation (SQuaRE). Product Quality Model*, ISO/IEC Std. 25 010:2023.
- [16] J.C. Laprie, “Dependable computing and fault-tolerance,” *Digest of Papers FTCS-15*, 1985, pp. 2–11.
- [17] J. de AG Saraiva, M.S. De França, S.C. Soares, J. Fernando Filho, and R.M. de Souza, “Classifying metrics for assessing object-oriented software maintainability: A family of metrics’ catalogs,” *Journal of Systems and Software*, Vol. 103, 2015, pp. 85–101.
- [18] I. Crnkovic, M. Larsson, and O. Preiss, “Concerning predictability in dependable component-based systems: Classification of quality attributes,” 2005.
- [19] S.S. Thapar, P. Singh, and S. Rani, “Challenges to development of standard software quality model,” *International Journal of Computer Applications*, Vol. 49, No. 10, 2012.
- [20] C. Izurieta, D. Reimanis, E. O’Donoghue, K. Liyanage, A.R.M. Muneza et al., “A generalized approach to the operationalization of Software Quality Models,” *PeerJ Computer Science*, Vol. 10, 2024, p. e2357.
- [21] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M.R. Chaudron, “A classification framework for software component models,” *IEEE Transactions on Software Engineering*, Vol. 37, No. 5, 2011, pp. 593–615.
- [22] M. Glinz, “On non-functional requirements,” in *15th IEEE International Requirements Engineering Conference*, 2007, pp. 21–26.
- [23] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, “Software quality models: Purposes, usage scenarios and requirements,” in *2009 ICSE Workshop on Software Quality*. IEEE, 2009, pp. 9–14.
- [24] N. Yilmaz and A.K. Tarhan, “Meta-models for software quality and its evaluation: A systematic literature review,” in *International Workshop on Software Measurement and the 15th International Conference on Software Process and Product Measurement, Mexico*, 2020.
- [25] M. Broy, F. Deissenboeck, and M. Pizka, “Demystifying maintainability,” in *Proceedings of the 2006 International Workshop on Software Quality*. ACM, 2006, pp. 21–26.
- [26] B. Kitchenham, “What’s up with software metrics?—a preliminary mapping study,” *Journal of Systems and Software*, Vol. 83, No. 1, 2010, pp. 37–51.
- [27] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A.E. Hassan, “How does context affect the distribution of software maintainability metrics?” in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 350–359.
- [28] B. Kitchenham and S. Pfleeger, “Software quality: The elusive target,” *IEEE Software*, Vol. 12, No. 9, 1996.
- [29] J. Eckhardt, A. Vogelsang, and D.M. Fernández, “Are ”non-functional” requirements really non-functional? an investigation of non-functional requirements in practice,” in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 832–842.
- [30] N.D. Anh, D.S. Cruzes, R. Conradi, M. Höst, X. Franch et al., “Collaborative resolution of requirements mismatches when adopting open source components,” in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2012, pp. 77–93.

- [31] J. Saraiva, S. Soares, and F. Castor, "Towards a catalog of object-oriented software maintainability metrics," in *2013 4th International Workshop on Emerging Trends in Software Metrics (WETSoM)*. IEEE, 2013, pp. 84–87.
- [32] C. Gencel, R. Heldal, and K. Lind, "On the relationship between different size measures in the software life cycle," in *2009 16th Asia-Pacific Software Engineering Conference*. IEEE, 2009, pp. 19–26.
- [33] C. Wohlin, E. Papatheocharous, J. Carlson, K. Petersen, E. Alégroth et al., "Towards evidence-based decision-making for identification and usage of assets in composite software: A research roadmap," *Journal of Software: Evolution and Process*, Vol. 33, No. 6, 2021, p. e2345.
- [34] B. Henderson-Sellers, "Bridging metamodels and ontologies in software engineering," *Journal of Systems and Software*, Vol. 84, No. 2, 2011, pp. 301–313.
- [35] G. Stănescu and S.V. Oprea, "Recent trends and insights in semantic web and ontology-driven knowledge representation across disciplines using topic modeling," *Electronics*, Vol. 14, No. 7, 2025, p. 1313.
- [36] C. Antoniou and N. Bassiliades, "A tool for requirements engineering using ontologies and boilerplates," *Automated Software Engineering*, Vol. 31, No. 1, 2024, p. 5.
- [37] N. Guarino, *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*, Vol. 46. IOS press, 1998.
- [38] D. Fensel, *Ontologies: A silver bullet for knowledge management and electronic-commerce*. Berlin: Springer-Verlag, 2004.
- [39] G. Van Heijst, A.T. Schreiber, and B.J. Wielinga, "Using explicit ontologies in KBS development," *International Journal of Human-Computer Studies*, Vol. 46, No. 2-3, 1997, pp. 183–292.
- [40] G. Guizzardi, M. Lopes, F. Baião, and R. Falbo, "On the importance of truly ontological distinctions for ontology representation languages: An industrial case study in the domain of oil and gas," in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2009, pp. 224–236.
- [41] G. Guizzardi, F. Baião, M. Lopes, and R. Falbo, "The role of foundational ontologies for domain ontology engineering: An industrial case study in the domain of oil and gas exploration and production," *International Journal of Information System Modeling and Design (IJISMD)*, Vol. 1, No. 2, 2010, pp. 1–22.
- [42] G. Guizzardi, "On ontology, ontologies, conceptualizations, modeling languages, and (meta) models," *Frontiers in Artificial Intelligence and Applications*, Vol. 155, 2007, p. 18.
- [43] G. Guizzardi, *Ontological foundations for structural conceptual models*, Ph.D. dissertation, University of Twente, 2005. [Online]. <https://research.utwente.nl/en/publications/ontological-foundations-for-structural-conceptual-models/>
- [44] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider, "Sweetening ontologies with dolce," in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2002, pp. 166–181.
- [45] M.F. Bertoa, A. Vallecillo, and F. García, "An ontology for software measurement," in *Ontologies for software engineering and software technology*. Springer, 2006, pp. 175–196.
- [46] J.R. Hilerá and L. Fernández-Sanz, "Developing domain-ontologies to improve software engineering knowledge," in *International Conference on Software Engineering Advances (ICSEA)*. IEEE, 2010, pp. 380–383.
- [47] M.J. Blas, S. Gomet, and H. Leone, "An ontology to document a quality scheme specification of a software product," *Expert Systems*, Vol. 34, No. 5, 2017, p. e12213.
- [48] N. Anquetil, K.M. de Oliveira, and M.G. Dias, "Software maintenance ontology," in *Ontologies for Software Engineering and Software Technology*. Springer, 2006, pp. 153–173.
- [49] O. Lassila and D. McGuinness, "The role of frame-based representation on the semantic web," *Linköping Electronic Articles in Computer and Information Science*, Vol. 6, No. 5, 2001, p. 2001.
- [50] N.F. Noy, D.L. McGuinness et al., "Ontology development 101: A guide to creating your first ontology," 2001.
- [51] W.A. Conklin, "Software assurance: The need for definitions," in *International Conference on System Sciences (HICSS)*. IEEE, 2011, pp. 1–7.

- [52] S. Sentilles, E. Papatheocharous, and F. Ciccozzi, “What do we know about software security evaluation? A preliminary study,” in *6th International Workshop on Quantitative Approaches to Software Quality, APSEC*, 2018.
- [53] S. Henninger and P. Ashokkumar, “An ontology-based metamodel for software patterns,” University of Nebraska – Lincoln, Tech. Rep. TR-UNL-CSE-2006-00005, 2006.
- [54] F. Béhé, S. Galland, N. Gaud, C. Nicolle, and A. Koukam, “An ontology-based metamodel for multiagent-based simulations,” *Simulation Modelling Practice and Theory*, Vol. 40, 2014, pp. 64–85.
- [55] S. Sentilles, E. Papatheocharous, F. Ciccozzi, and K. Petersen, “Property models for the automotive domain,” Mälardalen University, Tech. Rep., 2016. [Online]. <http://www.es.mdh.se/publications/4295->
- [56] J. Brank, M. Grobelnik, and D. Mladenić, “A survey of ontology evaluation techniques,” in *Proceedings of the Conference on Data Mining and Data Warehouses*, 2005, pp. 166–170.
- [57] A. Burton-Jones, V.C. Storey, V. Sugumaran, and P. Ahluwalia, “A semiotic metrics suite for assessing the quality of ontologies,” *Data and Knowledge Engineering*, Vol. 55, No. 1, 2005, pp. 84–102.
- [58] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, “A model for technology transfer in practice,” *IEEE Software*, Vol. 23, No. 6, 2006, pp. 88–95. [Online]. <https://doi.org/10.1109/MS.2006.147>
- [59] S. Sentilles, E. Papatheocharous, and F. Ciccozzi, “What do we know about software security evaluation? a preliminary study,” in *6th International Workshop on Quantitative Approaches to Software Quality*, 2018. [Online]. <http://www.es.mdh.se/publications/5277->
- [60] J. Maxwell, “Understanding and validity in qualitative research,” *Harvard Educational Review*, Vol. 62, No. 3, 1992, pp. 279–301.
- [61] K. Petersen and C. Gencel, “Worldviews, research methods, and their relationship to validity in empirical software engineering research,” in *Joint Conference of the 23rd International Workshop on Software Measurement and the Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA)*. IEEE, 2013, pp. 81–89.
- [62] J. Axelsson, “Evolutionary architecting of embedded automotive product lines: An industrial case study,” in *European Conference on Software Architecture*, 2009, pp. 101–110.
- [63] H. Gustavsson and U. Eklund, “Architecting automotive product lines: Industrial practice,” in *Software Product Lines: Going Beyond*. Springer, 2010, pp. 92–105.
- [64] R.A. McGee, U. Eklund, and M. Lundin, “Stakeholder identification and quality attribute prioritization for a global Vehicle Control System,” in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. ACM, 2010, pp. 43–48.
- [65] U. Eklund and J. Bosch, “Architecture for embedded open software ecosystems,” *Journal of Systems and Software*, Vol. 92, 2014, pp. 128–142.
- [66] M. Broy, “Challenges in automotive software engineering,” in *Proceedings of the 28th International Conference on Software Engineering, ICSE '06*. New York, NY, USA: ACM, 2006, pp. 33–42.
- [67] A. Pretschner, M. Broy, I.H. Kruger, and T. Stauner, “Software engineering for automotive systems: A roadmap,” in *Future of Software Engineering, FOSE '07*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 55–71.
- [68] R. Etemaadi, K. Lind, R. Heldal, and M.R. Chaudron, “Quality-driven optimization of system architecture: Industrial case study on an automotive sub-system,” *Journal of Systems and Software*, Vol. 86, No. 10, 2013, pp. 2559–2573.
- [69] C. Ebert and C. Jones, “Embedded software: Facts, figures, and future,” *Computer*, Vol. 42, No. 4, 2009, pp. 42–52.
- [70] B.W. Boehm, *Software Engineering Economics*, 1st ed. Prentice Hall PTR, 1981.
- [71] J. Li, G. Ruhe, A. Al-Emran, and M.M. Richter, “A flexible method for software effort estimation by analogy,” *Empirical Software Engineering*, Vol. 12, No. 1, 2007, pp. 65–106.
- [72] M. Jørgensen, “A review of studies on expert estimation of software development effort,” *Journal of Systems and Software*, Vol. 70, No. 1–2, 2004, pp. 37–60.

- [73] S.L. Pfleeger, F. Wu, and R. Lewis, “Software cost estimation and sizing methods: issues, and guidelines, Volume 269, Rand Comporation,” <http://www.rand.org/content/dam/rand/pubs/monographs/2005/RAND/MG269.pdf>, 2005.
- [74] V. Chapurlat, B. Kamsu-Foguem, and F. Prunet, “Enterprise model verification and validation: An approach.” *Annual Reviews in Control*, Vol. 27, No. 2, 2003, pp. 185–197.
- [75] R.S. Pressman, *A manager’s guide to software engineering*. McGraw-Hill, Inc., 1993.
- [76] R. Broek, J. Gorman, O. Haugen, G. Melby, B. Moller-Pedersen et al., “Quality by construction exemplified by TIME-the integrated methodology,” *Teletronikk*, Vol. 95, No. 1, 1999, pp. 73–82.
- [77] R. Morgan, *Component library retrieval using property models*, Ph.D. dissertation, Durham University, 1991.
- [78] M. Belguidoum and F. Dagnat, “Dependability in software component deployment,” in *Dependability of Computer Systems*, 2007, pp. 223–230.
- [79] A. Billig, S. Busse, A. Leicher, and J.G. Süß, “Platform independent model transformation based on triple,” in *Middleware 2004*. Springer, 2004, pp. 493–511.
- [80] V.A. Paun, *Precise and Adaptable Worst-Case Execution Time Estimation in Hard Real-Time Systems*, Ph.D. dissertation, Ecole Doctorale Polytechnique, 2014.
- [81] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner, “Measurement-based worst-case execution time analysis,” in *Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS’05)*, 2005, pp. 7–10.
- [82] A. Ermedahl, *A modular tool architecture for worst-case execution time analysis*, Ph.D. dissertation, Acta Universitatis Upsaliensis, 2003.
- [83] S. Sentilles, F. Ciccozzi, and E. Papatheocharous, “PROMOpedia: A web-content management-based encyclopedia of software property models,” in *40th International Conference on Software Engineering – Demonstrations track, ICSE*, 2018. [Online]. <http://www.es.mdh.se/publications/5086->

## Authors and affiliations

Efi Papatheocharous

e-mail: [efi.papatheocharous@ri.se](mailto:efi.papatheocharous@ri.se)

ORCID: <https://orcid.org/0000-0002-5157-8131>

Digital Systems, RISE Research Institutes of Sweden, Sweden

Séverine Sentilles

e-mail: [severine.sentilles@mdu.se](mailto:severine.sentilles@mdu.se)

ORCID: <https://orcid.org/00000-0003-0165-3743>

School of Innovation, Design and Engineering, Mälardalen University, Sweden

Kai Petersen

e-mail: [kai.petersen@hs-flensburg.de](mailto:kai.petersen@hs-flensburg.de)

ORCID: <https://orcid.org/0000-0002-1532-8223>

University of Applied Sciences Flensburg, Germany

Federico Ciccozzi

e-mail: [federico.ciccozzi@mdu.se](mailto:federico.ciccozzi@mdu.se)

ORCID: <https://orcid.org/0000-0002-0401-1036>

School of Innovation, Design and Engineering, Mälardalen University, Sweden