The present work was submitted to
the Research Group
Software Construction

of the Faculty of Mathematics,
Computer Science, and
Natural Sciences

Bachelor Thesis

Human–enabled management of software quality ontologies

presented by

Moritz Konstantin Rickert

Aachen, September 30, 2022

Examiner

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisor

Alex Sabau, M. Sc.

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Rickert, Moritz Konstantin — 407755

Name, Vorname/Last Name, First Name — Matrikelnummer (freiwillige Angabe) / Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit~~/Bachelorarbeit/ ~~Masterarbeit~~* mit dem Titel

I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

Human–enabled management of software quality ontologies

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, September 30, 2022 — _____

Ort, Datum/City, Date — Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Aachen, September 30, 2022 — _____

Ort, Datum/City, Date — Unterschrift/Signature

Acknowledgment

First and foremost, I would like to thank Prof. Dr. rer. nat. Horst Lichter for the opportunity to write my thesis at his chair and Prof. Dr. rer. nat. Bernhard Rumpe for reviewing my thesis.

For their part in my evaluation, I would also like to thank all participants for taking their time to evaluate the concepts of my thesis and for the feedback they provided. Without their participation, I would not have been able to validate my concepts statistically.

In addition, I would like to thank my family and friends for their constant support and proofreading of my thesis.

Last but not least, a special thank you goes to my supervisor M. Sc. Alex Sabau, for his guidance with direction and a great deal of engagement. His constructive criticism challenged my assumptions and understanding.

Moritz Konstantin Rickert

Abstract

Software quality is crucial in many areas of application. Therefore, asserting software quality is an essential activity of software engineering and research in this domain. *Software Quality Ontologies (SQOs)* formalize the complex and abstract components and semantic relationships of software quality for knowledge sharing by modeling semantic relationships between software quality concepts. However, due to the high complexity of SQOs, visual representation is beneficial. Up to our knowledge there does not exist an ontology management tool supporting the unique needs to represent the characteristics of SQOs.

This thesis introduces a new approach for the visual representation and maintenance of SQOs by reconciling different representation techniques. We evaluate our approach by conducting a user survey with the System Usability Scale. The quantitative analysis of our concepts concludes that the presented approach statistically significantly improves the user experience. In addition, the qualitative analysis of the user survey reveals a better understanding of software quality.

Utilizing our concepts in SoQOnto, we are able to support researchers in their understanding and maintenance of SQOs. Additionally, the extendability and replaceability of SoQOnto allows for the implementation of new concepts not considered in this thesis.

Contents

List of Tables

List of Figures

List of Source Codes

1. Introduction

We depend on software in nearly every aspect of our lives, e.g., at our workplace or even in life-threatening situations at a hospital. Even if we only consider the economic fallout of non–reliable software, the costs are enormous. In 2020, in the United States alone, software failing in operation cost companies an estimated $1.56 trillion [Kra21]. Therefore, we need reliable software to reduce costs and ease our everyday lives.

As a result, software quality is an important research field of software engineering [KLR16]. Assuring software quality means taking actions necessary to comply with given quality requirements [90]. Defining these requirements needs an understanding of software quality and its measures [Bøe08]. Measurability and comparability of software quality are described and structured by software quality models. The most used software quality model is SQuaRE (ISO/IEC 25010) [11; HP16]. Understanding the effects of measures contained in software quality models on each other is an essential aspect of understanding software quality [Bøe08].

To formalize these correlations, complementing the taxonomy of software quality models allows for knowledge sharing [SBF98]. *Software Quality Ontologies (SQOs)* extend the taxonomy of software quality found in software quality models with a formalization of semantic relations, including correlations [Sun+20]. As software quality is an extensive, abstract domain, numerous components and relations are included [MD06; HP16]. Grasping information at this scale is a hard–to–solve problem, since humans have a limited ability to fathom complex problems [LL13]. Even on problems with comparatively low complexity, we draw unsound conclusions. Visualizations aid the human perception and exploration of data, allowing us to understand more complex problems [Reu+90]. Therefore, to complement the researchers' skills in SQO management, we aim to introduce a visual representation of the complex field of software quality.

In this thesis, we aim to enhance the human–enabled management of SQOs through the introduction and combination of representation and management methods. We utilize well–proven ontology representation methods and newly introduced methods for the representation and management of SQOs. Using an ontology task classification [NSL18], we subdivide our effort into representations targeted at each task class. For decreased abstractness, we create a real–world ontology incorporating characteristics of SQOs. The real–world ontology is based on the field of a car as an everyday item for the familiarity of the artifacts contained. With this easier–to–understand ontology, the focus can be laid on the before–mentioned methods implemented in our web application SoQOnto. In the next section, we outline the structure of this thesis.

1.1. Structure of This Thesis

We begin by introducing background information vital to understanding our research in chapter 2. In chapter 3, we provide the motivation and problem statement, including our research questions and the research method for this thesis. Then, in chapter 4, we introduce related work and argue their relation to SQOs. Beginning with our work, we describe the backgrounds of, and the concepts for SQO representation and management in chapter 5. Following our concepts, we outline our application's design in chapter 6. Then, in chapter 7, we introduce our realization and evaluate, and discuss the results of our evaluation conducted as a user survey in chapter 8. Finally, we conclude this thesis by summarizing our research findings and outlining possible future work in chapter 9.

2. Background

Contents

In this chapter, we introduce background knowledge needed in the course of this thesis. We begin by introducing ontologies and SQOs as the data source of our visual representation. In the following section, we introduce the Resource Description Framework as the data exchange model for ontologies, the Web Ontology Language as a vocabulary extension, and the textual syntax for *Resource Description Framework (RDF)* definitions Turtle, which we use in this thesis.

2.1. Ontologies

The word "ontology" has its roots in the philosophical sense of the research of attributes belonging to something because of its existence [GOS09]. Philosophically, ontologies emerged in the field of *nature* and its *structure*. In this context, the physical existence of the concept to be represented is of no concern.

With the emergence of knowledge engineering, ontologies found their way into computer science. In the knowledge engineering community, ontologies define computational artifacts [GOS09]. We utilize Studer et al.'s definition of ontologies in the field of computer science as *"a formal, explicit specification of a shared conceptualization"* [SBF98]. This definition is based on both Gruber's definition as an "explicit specification of a conceptualization" [Gru93] and Borst's definition as a "formal specification of a shared conceptualization" [BAT97]. All definitions refer to a *conceptualization*, underlining that ontologies are abstract models of reality. Additionally, Studer's definition includes the aspect of a *shared conceptualization*, hinting at the important use case for knowledge sharing. Ontologies structure this shared knowledge through *concepts* and *relations*. An ontology combines the hierarchical structure (a *taxonomy*) with non-hierarchical relations. [GOS09]

2.2. Software Quality Ontologies

Essential instruments in describing and structuring software quality are software quality models. The included structural concepts and relations start from the top down with *Quality Attributes*, which subdivide software quality into clearly defined parts. These Quality Attributes are abstract and are subdivided into *Quality Factors*. Quality Factors are measured with Metrics. Thus, in order to evaluate compliance with Quality Attributes, the Metrics of Quality Attributes must be determined and measured. The collected values can then be indirectly assigned to the Quality Attributes via its Quality Factors.

When researching software quality, we need a well–formalized representation of the software quality model's concepts. To address not only the structure of software quality but also the semantical relations, including correlations, software quality models need an extension. Therefore, we define SQOs as follows:

Definition. *A Software Quality Ontology formalizes the concepts and relations of software quality for shared conceptualization. It complements the concepts and hierarchy found in software quality models with additional semantically associated relations.*

In this thesis, the semantically associated relations are restricted to measure–to–measure correlations and correlations in parallel to hierarchical relations.

2.3. Resource Description Framework

To formally define the concepts contained in SQOs, we use the Resource Description Framework. The RDF [GS14] is a web standard first defined by the *World Wide Web Consortium (W3)* in 1996, outlining a standardized model for data exchange on the world wide web. Although the W3 designed the RDF for describing web content, resources do not need to be accessible on the web. This design allows describing nearly everything[1] in a standardized way. [McB04]

Resource descriptions in the RDF consist of named properties and their values [Bec14]. The predefined vocabulary in *RDF Schema (RDFS)* [BG14] can be used to define properties, classes, and relationships. Many ontologies use aspects of RDFS. Used aspects include classes (*rdfs:Class*) and subclass relationships (*rdfs:subClassOf*). Additionally, when defining a property, constraints on the subject's and object's class membership are defined using *rdfs:domain* and *rdfs:range*, respectively.

To uniquely identify the resources defined, a *URI* is used. The components of a URI are the *prefix*, identifying the resource as part of a vocabulary (e.g., for rdfs: "http://www.w3.org/2000/01/rdf-schema#")[2], and a name in that vocabulary. However, some resources named *blank Nodes* or *b-Nodes* do not have a URI. Vocabularies may include blank Nodes since RDF statements can only formalize relationships between

[1] RDF does not include negations and universal quantifiers

[2] For simplicity in the course of this chapter, we abbreviate the URI with the short-form "*prefix:name*" (e.g., using "*rdfs*" as a prefix rather than the absolute URI).

two Nodes. Blank Nodes provide a possibility of structuring data. However, as RDF's background does not lie in the space of ontologies, essential components of ontologies are not defined. [McB04]

2.3.1. Web Ontology Language

The *Web Ontology Language (OWL)*[1] [MH04], also defined by the W3, extends the RDF with additional ontology–specific vocabulary, allowing for simpler ontology definitions. Since the OWL's background lies in the field of ontologies, it provides vocabulary depended on in many ontologies defined using the RDF. The OWL's vocabulary targets the representation of knowledge divided into concepts, groups of concepts, and their relations (e.g., domain-dependent range definitions for properties). Therefore, the OWL allows the definition of entire ontologies in the RDF. [AH04; MH04]

2.3.2. Turtle

To define concepts and relations of SQOs in the RDF, we use the *Terse RDF Triple Language (Turtle)*. Turtle is a compact form of writing RDF syntax and creating an RDF graph in natural text form, compatible with the N-Triples format published with the RDF [Bec14]. Turtle introduces abbreviations for datatypes and commonly used patterns, significantly shortening the document size of ontologies and increasing readability. One abbreviation significantly improving the readability of an ontology is the *prefix* definition previously used in this chapter for readability. A comparison of the readability using prefixes and the *a* abbreviation (rdf:type) is shown in source code 2.1 and source code 2.2. For this reason, we use Turtle when referencing RDF definitions in this thesis.

```
1  <http://example.com/#HumanEnabledMgmtOfSQOs> rdf:type <http://example.
   com/#Thesis> ;
2      <http://example.com/#ThesisTitle> "Human-enabled management of
   software quality ontologies" ;
3      <http://example.com/#ThesisStudent> "Moritz Konstantin Rickert" ;
4      <http://example.com/#ThesisChair> <http://example.com/#SWC> ;
5      <http://example.com/#ThesisSupervisor> <http://example.com/#
   AlexSabau> ;
6      <http://example.com/#ThesisExaminer> <http://example.com/#
   HorstLichter> ;
7      <http://example.com/#ThesisExaminer> <http://example.com/#
   BernhardRumpe> .
```

Source Code 2.1: Turtle definition of this thesis without prefix usage

[1] http://www.w3.org/2002/07/owl

```
@prefix ex: <http://example.com/#> .

ex:HumanEnabledMgmtOfSQOs a ex:Thesis ;
    ex:ThesisTitle "Human-enabled management of software quality
ontologies" ;
    ex:ThesisStudent "Moritz Konstantin Rickert" ;
    ex:ThesisChair ex:SWC ;
    ex:ThesisSupervisor ex:AlexSabau ;
    ex:ThesisExaminer ex:HorstLichter ;
    ex:ThesisExaminer ex:BernhardRumpe .
```

Source Code 2.2: Turtle definition of this thesis with prefix usage

3. Motivation and Problem Statement

Software quality is a much–researched area of software engineering [Boe+07]. As a result, structured knowledge about the characteristics contained in software quality models is constantly being extended [11; HP16]. To formalize these characteristics and their semantical relations and allow for knowledge sharing, SQOs are employed [Sun+20; SBF98]. However, quality attributes have an influence on other quality attributes [HP16], therefore, increasing the complexity of already complex and abstract characteristics of SQOs [MD06]. In our view, because of these correlations and the static nature of representation, existing solutions, like Protégé[1], offer insufficient support for understanding software quality. Therefore, the need for dedicated representation concepts focused on the specific characteristics of SQOs arises.

The abstract nature of software quality prohibits a focus on the concepts of ontology management and representation, as the complex data contained needs understanding. To enable the introduction of concepts, we formalize a domain of an everyday item with reduced abstractness in a real–world ontology. This leads us to our first research question:

> **RQ1:** *How can a real–world ontology, incorporating all relevant aspects of a Software Quality Ontology, look like?*

Even with a real–world ontology providing reduced abstractness, a software–based visualization needs concepts to represent a large amount of information. However, since, to our knowledge, no tool targeting the representation of SQOs exists, most concepts employed must prove themselves in practice, leading us to our second research question:

> **RQ2:** *How can a visual representation support the understanding of the components and relations of SQOs?*

We can differentiate the ontology representation concepts into two groups. Firstly, concepts that represent the effect of correlations between two attributes and, secondly, concepts that distinguish between different characteristics of SQOs. The importance of understandability of correlations in SQOs is underlined through the amount of jointly used measures in software quality models (See table 3.1). To reflect the groups of concepts needed, we also divide our research question:

> **RQ2.1:** *How can the effects of Correlations between Measures and Metrics be visualized?*

[1] https://protege.stanford.edu/

RQ2.2: *How can the aspects of quality be represented in an understandable way?*

With the use of these two groups of concepts for a better understanding of software quality, the second principle of ontologies, the shareability of knowledge, and additionally management, still need to be implemented:

RQ3: *How can the modification and sharing of a represented ontology be simplified?*

In order to answer these questions, we introduce our research method and real–world ontology in the next section.

	Compatibility	Portability	Maintainability	Performance Efficiency	Security	Usability	Reliability
Functional Suitability	3	3	3	2	2	2	2
Reliability	3	3	5	2	3	2	
Usability	2	2	2	2	1		
Security	2	2	3	2			
Performance Efficiency	3	2	3				
Maintainability	4	3					
Portability	3						

Table 3.1.: Amount of Measures jointly used for Quality Attributes based on [HP16; 11]

3.1. Research Method

This section introduces our real–world ontology as the research method we apply during the development of our concepts and *SoQOnto*. To generate results faster and extend our ontology and tool simultaneously as needed, we opt for an iterative approach, utilizing the concept of *rapid prototyping* [Gra94]. Our first prototypes are developed using a top–down approach [Cha91], enabling us to divide–and–conquer the characteristics we see and suspect in SQOs. For later prototypes incorporating additional characteristics, the approach is changed to a bottom–up approach [Jør04]. Using the bottom–up approach allows us to sharpen both the prototypes of ontology and software by gaining an increased understanding of the project's execution and planning [Jør04]. We then evaluate the effectiveness of newly introduced prototypes and only adopt concepts that effectively support the users with their tasks. Prototyping is a perfect fit for our needs since not all concept requirements can be determined prior to development [LL13]. Gordon et al. [GB95] also found the rapid prototyping method to help create software that is

easier to use and better tailored to the users' needs, therefore, enabling humans to manage SQOs. Meanwhile, the effort needed for a working product decreases. Based on this approach, we introduce the specifics of our real–world ontology used during development in the remainder of this section.

3.1.1. Real–World Ontology

In order to begin the process of prototyping visualization concepts for SQOs, we need an ontology incorporating all characteristics expected in SQOs. We introduce a real–world ontology for developing representation concepts to avoid the need for time–consuming familiarization with the topic of software quality. Therefore, we base our real–world ontology around an everyday item: the "Car." Using car vocabulary in our ontology allows us to implement the characteristics of SQOs, while not worrying about technicalities and abstractness of software quality attributes. In the spirit of rapid prototyping, we create a small basic ontology framework with only a few Attributes, which we extend as we develop further concepts as needed.

Since the "Car" ontology development starts before the implementation of *SoQOnto* – needing a visualization – we develop the first iterations using the online tool *draw.io*.[1] We begin by defining a taxonomy containing only Attributes (not Quality Attributes) and Metrics. Afterwards, amending a minimal amount of Correlations incorporating Attributes, and Metrics as classes and Subclasses (See chapter 2.3), and Correlations as relationships (figure 3.1), concluding the development conducted using a top–down approach. For the incorporation of additional concepts, the first prototype of SoQOnto is available. Therefore, this iteration concludes the ontology development conducted outside of SoQOnto.

During the implementation using SoQOnto and the bottom–up approach, it becomes apparent that while all characteristics of SQOs are found within the domain of a car, as it can contain the aspect of car *quality*, it differs structurally from SQOs through its broad domain and many independent parts. To narrow the represented field and gain a structurally more similar real–world ontology, a car quality ontology could be used.

[1] https://app.diagrams.net/

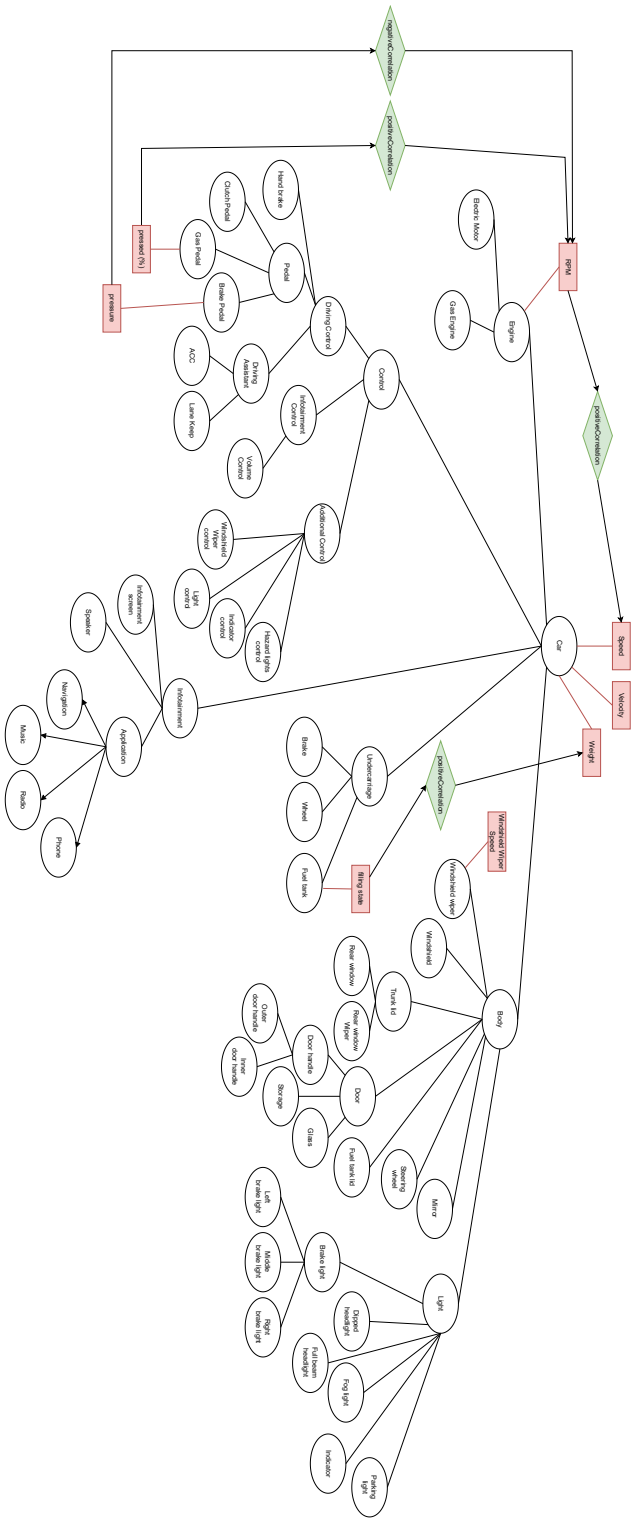


Figure 3.1.: Last iteration of the "Car" ontology before development of SoQOnto

4. Related Work

Contents

In the previous chapter, we considered the motivation and questions we want to answer with our concepts. This chapter outlays the related work regarding the extraction of smaller, more manageable sub-ontologies of SQOs, then introduces existing tools usable for SQOs visualization and management.

4.1. Sub-Ontology Extraction

Ontologies often reach sizes that are no longer manageable. Hence, it is essential to be able to compute and display sub-ontologies. In this section, we describe approaches to compute sub-ontologies of target Nodes. Finally, we distinguish our work from the presented approaches.

4.1.1. Extracting Superclass-Based Sub-Ontologies

A simple, reasoner-based, sub-ontology extraction is introduced in [El +20]. In their paper, El Bolock et al. extract an ontology's sub-ontologies by computing all superclasses contained in the ontology with a reasoner. The reasoner then adds all subclasses and properties of the superclasses' entities to a new ontology. This sub-ontology extraction method introduces one sub-ontology for each superclass and overview ontology referencing all sub-ontologies.

4.1.2. Clustering

Clustering introduces a lower-dimensional representation of high-dimensional ontologies. In their work, Shaik et al. [Sha+06] apply clustering on relations to achieve representation in a three-dimensional space. The clustering process decides which relations it combines into a cluster using explicitly defined semantics of the relations. The *Relation Semantics Elicitation Prototype* [KRT06] introduces the semantics used. Once computed, the clustering method returns its result in textual format since no appealing visual representation of a 32-dimensional space exists.

4.1.3. Sub-Ontology Extraction by Hierarchical Traversal

Another method of extracting sub-ontologies is traversing the target Node's hierarchy. This process includes the target Node's superclass and their superclass until reaching the topmost Node. Similarly, the same process computes all subclasses. Additionally, the result includes linked (by non-hierarchical relations) Nodes and their superclasses. [SR06]

4.1.4. Sub-Ontology Extraction Using Hyponym and Hypernym Closure

The approach of extracting a sub-ontology by hyponym and hypernym closure, as introduced by Ranwez et al. in [RRJ11], differs from the hierarchical traversal approach as it stops traversal at the *greatest common descendant (gcd)* and *least common ancestor (lca)*. In order to use *gcd* and *lca*, the examined sub-graph must be acyclic. Therefore, the set of relations considered is limited to *is-a* or *Subclass* relations as these are acyclic. The approach extends the set of relevant concepts with non-hierarchical relationships. It only considers transitive relations. Connecting Nodes in between are no part of the relevant concepts.

4.1.5. Conclusion

None of the sub-ontology extraction methods are suitable for Software Quality Ontologies. Some restrict the types of ontology they can handle. Others cannot be displayed visually appealing. Without visual support, human-enabled management is impossible. Alternatively, they introduce rigid boundaries around ontology parts. These restrictions reduce the possibilities for users to track interrelations between Quality Attributes and Metrics. Lastly, Seidenberg et al.'s approach does not include subclasses of linked Nodes because otherwise, it would include the whole ontology. However, in Software Quality Ontologies, most linked Nodes not already included (correlations from measure to measure) are the bottom-most Nodes of the taxonomy. Hence, this results in a large sub-ontology. In this thesis, we will address the identified gaps through our concepts introduced for SQOs. In the next section, we discuss the limitations for SQOs of existing tools usable for ontology management.

4.2. Related Tools

Due to the graph-based nature of ontologies, there are a variety of tools that could be used to develop and manage SQOs. In this section, we address tools in order from generic graph creation tools to ontology management tools. We conclude this section by summarizing the common limitations of the tools presented.

4.2.1. draw.io

Draw.io is a free, open-source web application for the generation of graphs and diagrams. Through its extensive scope, it is used for many different requirements [JGr22]. For example, flowcharts, UML diagrams, and network diagrams can be realized. Figure 3.1 shows an iteration of our real–world ontology consisting of hierarchically structured Attributes and Correlations. Every graph realized in draw.io is static. Hence, the user has to adjust the graph's layout when amending the SQOs manually, and the effects of correlations cannot be represented reactively.

4.2.2. Neo4j

As a graph database, *Neo4j* solves draw.io's issue of graph layouting. Through the Neo4j browser, users can visualize and interact with the ontology in a graph format [neob]. In contrast to draw.io, Neo4j does not offer UI–supported creation, editing, or removal of Nodes and relations. To modify a SQO, users have to learn Cypher, Neo4j's query language, offering detailed ontology part selection possibilities [neoa]. Like draw.io, Neo4j does not offer a reactive visual representation of the effects of correlations.

4.2.3. Protégé

As the last tool introduced, *Protégé* is the most widely used software for ontology editing [Mus15]. It offers an extensive UI for creating, editing or removing Nodes and relations. As it stands, we do not know of any visualization for Protégé visually supporting the effects of correlations.

4.2.4. Conclusion

All presented tools, from generic graph to ontology editor, have their background in the applicability for a large domain of use-cases, all based on static data. As a result, no tool supports the visual representation of the effects of correlations crucial for the understanding of interdependencies in SQOs. In the next chapter, we introduce our concepts to support the understanding of these interdependencies.

5. Concept

Contents

With this chapter, we want to introduce concepts solving the difficulties users face when trying to understand and manage the components of SQOs. To achieve this goal, we begin with the introduction of the most used software quality model and research extending it. This allows us to outline and contextualize the components of SQOs, which we also need to incorporate in the iterations of our real–world ontology. As discussed in chapter 3, representation is needed to support the understanding of SQOs. We then introduce existing methods of multivariate graph representation. Later, we adapt these for the representation of SQO components. This adaption process is led by the differentiation between two classes of ontology tasks: *overview* and *focus tasks*. Hence, as the last part of our concept's background, an ontology task classification is introduced.

Using the insights provided by this chapter's background, we define the components of SQOs, aiming at a shared understanding. Only then do we turn to introducing our concepts for representing and managing SQO components.

5.1. Conceptual Background

In this section, we use the ISO/IEC 25010 software quality model and research based on it to outline the components commonly found in SQOs. We then introduce multivariate graph representation techniques. In the last part of this section, ontology tasks are classified into *focus* and *overview tasks*. Our concepts, introduced in the next section, accommodate for the components and task requirements of SQOs by adapting and

extending the representation techniques outlined in this section.

5.1.1. Software Quality Model

SQuaRE (ISO/IEC 25010)

The ISO/IEC 25010 standard is the most used software quality model and forms the basis of many SQOs [HP16]. It provides an overview of components of software product quality, containing eight top-level components (Quality Attributes) divided into second-level components (Quality Factors). For example, the Quality Attribute "Security" is divided into the Quality Factors "Confidentiality," "Integrity," "Non-repudiation," "Authenticity," and "Accountability." A combination of Quality Factors defines the degree of fulfillment to the Quality Attribute. Each Quality Factor's degree of fulfillment is, in turn, a composition of several Measures' values [HP16]. In the following research, an extension of the ISO/IEC 25010 taxonomy to an ontology is motivated. [11]

Mutual Influences in ISO/IEC 25010

In ISO/IEC 25010, various Measures' effects influence multiple Quality Attributes' values. In their paper, Hovorushchenko et al. [HP16] discuss this mutual influence of Measures on Quality Attributes. With their introduction of weights for each Measure, they differentiate Measures whose value affects multiple Quality Attributes from others, extending the ISO/IEC 25010 taxonomy to an ontology. They deduce that the presence of values of Measures that affect more Quality Attributes is more critical when deriving software quality. These effects motivate the need for specialized SQO representation. In combination, the research introduced outlines most SQO components we want to visually represent.

5.1.2. Multivariate Graph Visualization Techniques

We use multivariate graph visualization techniques to represent the SQO components extracted in the previous section. As in ontologies, a crucial defining factor of multivariate graphs is the presence of additional data on both Nodes and relations [KPW14]. Using these techniques, SQO component information and differentiation can be visually represented. In their paper, Partl et al. [Par+12] introduce four different techniques used in multivariate graph visualization. In the following, we outline the three techniques applicable for SQO representation we adapt, combine, and extend in our concepts.

Layout Adaption

Layout adaption introduces the ability to change the layout based on the amount and types of Attributes to be visualized. The wide range of layout adaption visualizations begins with scatter plots that visualize the relation between two Attributes [Bez+10] to a table-based visualization containing a column for each Attribute contained. [Par+12]

Separate Linked Views

The visualization technique of linking multiple separate views introduces multiple viewing points on the same Nodes and Attributes, as used for multivariate graph drawing in [SHQ08]. Applying this technique allows combining the different advantages of multiple views. However, linking information between views requires high user interaction, increasing the complexity of the information representation. [Par+12]

On-Node Mapping

On-Node mapping is a widely used method of showcasing information on Nodes in a graph-based environment using visual clues on the Nodes themselves. Visual clues can include color, form, glyphs, and scaling. Combining the visual clues with the information they represent requires user interaction. Therefore, on-Node mapping is best used with tasks requiring a topological view of the graph. Which tasks require topological information is classified in the next section. [Par+12]

5.1.3. Classification of Ontology Tasks

As introduced by Nobre et al. in [NSL18] working with ontologies can be divided into two classes of tasks: *focus* and *overview tasks*. To support human–enabled management, the selected representation concepts need to focus on the different tasks researchers perform on SQOs. For example, gaining an overview or finding all correlated measures affecting the value of a measure. For a more detailed classification, consult the work of Lee et al. [Lee+06].

The term *focus task* relates to tasks in which the details and relations of a focussed Node are essential. For focus tasks, the global structure of the ontology is incidental. Examples of focus tasks include analyzing a Node's neighborhood and its details, the relation type, accessing information of related Nodes, and more. [NSL18]

Overview tasks are complementary to focus tasks. To complete given overview tasks, the user needs information about an ontology's global structure and topology [NSL18].

We now have possible partial representations for the introduced SQO components to be targeted at the task classification. For a shared understanding, we formally define the components of SQOs in the next section.

5.2. Component Definitions

As we will discuss the different aspects of SQOs both in the context of our implementation as well as the data structure itself in the further course of this thesis, we find it essential to distinguish and formally define the different terms used in those separate contexts. For comprehensible examples, we choose artifacts from our car ontology in this section.

Node

A *Node* is an entity representing a specific part of an ontology, e.g., an "engine." It can contain additional information associated with it by Arcs as defined in the RDF[1] [GS14]. Node is the hypernym for components extending the Node term. These components are mutually exclusive subsets.

Arc

An *Arc* defines a predicate relation between two Nodes (N), e.g., a subclass Arc between "Infotainment" and "Navigation," resulting in the triples used to define an ontology in the RDF [GS14].

$$A = \{(n, m) \mid \text{exists an Arc from n to m}, n \in N, m \in N\}$$

Attribute

In the context of SQOs, we refer to the most basic subset of Nodes as *Attributes*. Most Nodes defining the ontology's taxonomy are Attributes as they define the main concepts and part relations.

Measure

We refer to a Node as a *Measure* when it represents raw data that can be directly consumed (e.g., "tire diameter") [Tea21]. The affiliation of each Measure is marked by a specific Arc (the "measureOf" Arc).

Metric

A Node is referred to as a *Metric* if it combines one or more Measures [Tea21]. The output of a Metric again is defined as a Measure. For example, by combining the Measure "tire diameter" with the Measure "rotations per minute," we can compute the Metric "current speed."

Quality Attribute

Quality Attributes are properties of a system or a system's behavior as perceived by the user, measured by Measures and Metrics, and described by their output (a Measure). The value of how well a Quality Attribute is satisfied in a given system is entirely subjective to each user's needs, based on trade-offs between different Measures. For example, a trade-off between "BHP" and "Miles per gallon." [Bar+95]

[1] https://www.w3.org/RDF/

Correlation

A *Correlation* is an Arc defining the impact of value changes in-between Measures. It can either be positive (a value increase increases the value of the connected Node) or negative (a value increase decreases the value of the connected Node). Each Correlation has an associated weight ($weight(n, m)$) defaulting to 1 describing the impact severity on the associated Node's value. Correlations also apply to Quality Attributes because they inherently influence each other as perceived by the user. However, because of the subjective nature of a Quality Attribute's value, only the existence of a Correlation can be indicated, not a specific value change.

$$corr_+ = \{(n, m) \mid (n, m) \in E, \text{ Arc is a positive Correlation}\}$$

$$corr_- = \{(n, m) \mid (n, m) \in E, \text{ Arc is a negative Correlation}\}$$

Improvement Tactic

Improvement Tactics are closely related to Quality Attributes and Measures and aim at increasing the Quality Attribute's degree of fulfillment by enhancing its Measures' values. Enhancing means increasing a value (e.g., "miles per gallon") or decreasing a value (e.g., "fuel consumption") based on the current context. Therefore, Improvement Tactics also correlate to Measures, suggesting whether an improvement is an increase or decrease in value. In this scenario, Improvement Tactics can be applied on a scale. Therefore, they need to have a value that represents their degree of fulfillment. This value closely depends on the correlated Measures' values, e.g., increasing the Metric "fuel consumption" would lead to a decrease in the value of an "efficient driving" Improvement Tactic.

5.3. Algorithms

This section uses the definition of SQO components to introduce algorithms essential for our representation concepts. To address the core defining characteristic of SQOs, a value calculation algorithm based on correlated Nodes is introduced. This algorithm lays the groundwork for a visual representation of the effects of interdependencies in SQOs. To address not only the characteristics of SQOs but also the specifics of the task class, we introduce a root node calculation algorithm. Focus tasks depend on a focused node. This algorithm provides the root node as a focused node, if otherwise none is set.

5.3.1. Value Calculation

We calculate the value of correlated Nodes on value change to be able to visually represent dependencies. When representing this value, users are enabled to understand the connections between artifacts of the ontology. Our value calculation algorithm respects

the given constraints and, most importantly, the weights of Correlations. Figure 5.1 outlines the value calculation algorithm on an abstract example. We explain the exact calculation depicted in the following.

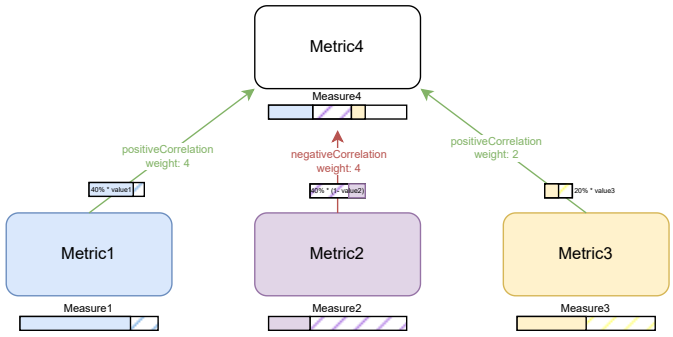


Figure 5.1.: An abstract example of the value calculation algorithm.

To explain the functionality of the value calculation algorithm, we introduce a set of *source Measures* (S_n) and a *target Measure* (t), where the target Measure is the Measure of which we are currently calculating the value. Source Measures are all Nodes correlated to the target Measure.

$$S_n = \{s \mid (s,t) \in corr_+ \vee (t,s) \in corr_+ \vee (s,t) \in corr_- \vee (t,s) \in corr_-\}$$

The target Measure's value ($val(t)$) then consists of a combination of all source Measures' values based on the type of Correlation ($val_t(s)$) and weight divided by the total weight of all correlated source Nodes.

$$val(t) = \left(\sum_{s \in S_n} weight(s,t) / (\sum_{l \in S_n} weight(l,t)) \right) * val_t(s)$$

A source Node's value on the target is adopted unchanged if the Correlation is positive. Suppose the Correlation is, however, negative. In that case, we use the remaining value of the target Node by calculating the difference between the maximum and current value of the source Node.

$$val_t(s) = \begin{cases} val(s) & \text{if the Correlation is positive} \\ 1 - val(s) & \text{else} \end{cases}$$

In our calculations, all Node values are a float between 0 and 1. Values displayed in the representation are either shown as a percentage, if no *minimal value* (v_{min}), *maximal value* (v_{max}), and *unit* (u) are specified, or are calculated based on the previously mentioned values:

$$\text{unitval}(s) = v_{min} + (v_{max} - v_{min}) * val(s)$$

In figure 5.2, we show an illustrative example of this algorithm. It illustrates the water temperature of a pot, influenced by the amount of cold and hot water added. In both depicted cases, both cold and hot water have a correlation weight of 1 to the pot temperature. Hence, each is effectively responsible for half of the resulting pot temperature. As the correlation between hot water and pot temperature is positive, the amount's value is directly adopted (illustrated in red). The cold water's correlation is, however, a negative one. Therefore, the difference between the maximal and the actual values is adopted. This difference is represented through blue stripes as the "Remaining Cold Water amount."

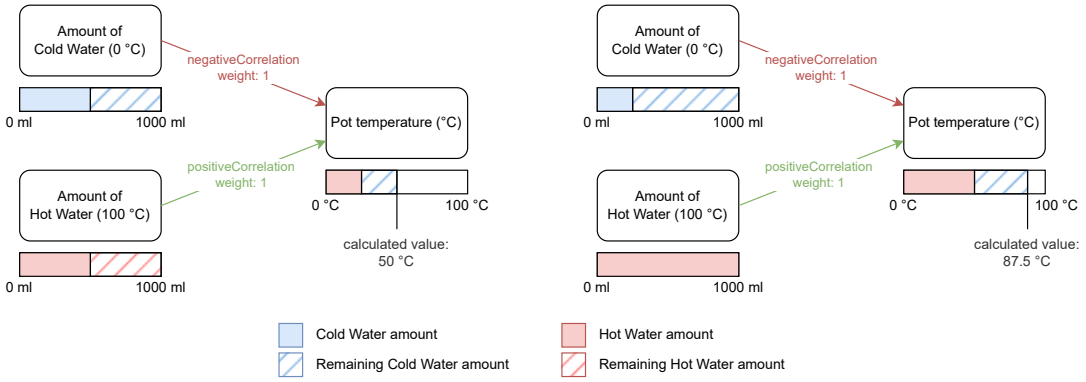


Figure 5.2.: A real-world example of the value calculation algorithm.

When one Node's value is changed, the value calculation algorithm is run for all Nodes directly or indirectly correlated to this Node. We opt for a round-based model to prevent infinite loops of value adjustments, as seen in figure 5.3. Starting with the triggering Node in the first round, the algorithm continues with all correlated nodes not part of previous rounds. Nodes part of previous rounds are ignored. When no new Node is added, the algorithm has calculated the new values for each Node exactly once.

5.3.2. Finding the Ontology's Root Node

Some methods of representing SQOs need a starting point. Whenever the user does not set this starting point, we use the ontology's root Node (r) calculated using the ontology's Nodes (N) set. However, parts of the ontology might not be connected (yet) in some cases. Unconnectedness can be due to the ontology being in the creation phase or by design. In this case, we want to find the root Node of the biggest subgraph.

Our calculation starts by finding the set of Nodes without an outgoing *subclass* relation R. If only one Node exists, the ontology has no unconnected parts, and we return the only possible root Node. Otherwise, we iteratively add all Nodes connected by a subclass

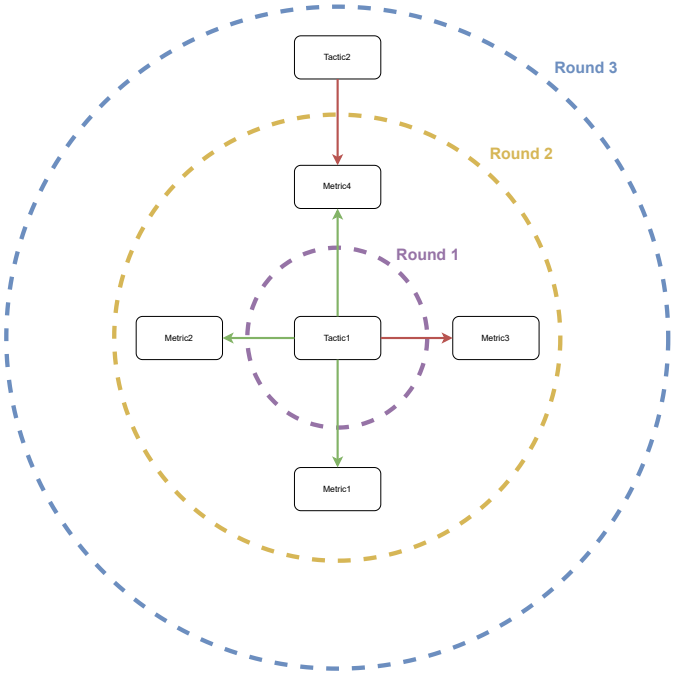


Figure 5.3.: An abstract view on the round-based value calculation triggered by a value change of Tactic1.

relation to the possible root Nodes. We call these Nodes *accessible Nodes*. We return the Node n ($n \in R$), with the most extensive set of accessible Nodes (A).

$$R = \{n \mid \forall m \in N : \neg\text{subclass}(n, m)\}$$

$$r = \begin{cases} n & \text{if } |N| == 1, n \in N \\ argmax\{|\text{accessible}(n)| \mid n \in N\} & \text{else} \end{cases}$$

$$\text{accessible}(n) = \{m \mid \text{subclass}(m, n) \vee \exists\ l : \text{subclass}(m, l) \wedge l \in \text{accessible}(n)\}$$

5.4. Ontology Representation Concepts

With the algorithms outlined in the previous section, the groundwork for human-enabled SQO representation is laid. In this section, we introduce our overall approach to representation concepts in the form of task-focused ontology representation. It divides the introduced concepts into concepts for overview and focus tasks. Two separate views include the corresponding concepts, addressing the different prerequisites of the classified tasks. Therefore, a view is a combination of concepts to create a combined representa-

tion. In order to not always display the complete ontology in the two views, we introduce filters extending the views. Filters remove further complexity from the represented ontology by hiding distant Nodes or components as a whole. We conclude this chapter with a concept to simplify the management by use of the human-enabled representation.

5.4.1. Task-Focused Ontology Representation

Based on the previous definition of overview and focus tasks described in chapter 5.1.3, we aim to introduce two different views based on the different needs resulting from these classes. The *overview view* incorporates representation concepts into a graph aimed at support for overview tasks. Meanwhile, the *treetable view* focuses on support for focus tasks, combining hierarchical information in the form of a tree (similar to a file tree) with additional attributes of the Node in columns of a table. This view is loosely based on the *tree+ table* view introduced in [NSL18].

To avoid tying users to one view, we use the concept of multiple linked views so that users can switch back and forth between views without having to replicate the context currently represented.

Overview View

To represent and distinguish SQO components in the *overview* view, we use on-Node mapping, introducing Node, and relation type differentiation by color and value differentiation of Measures, Metrics, and Improvement Tactics based on Node size.

As the overview view's purpose includes showcasing the topology of the complete ontology, a graph representation allows a complete view of the ontology and its Node and relation types. However, this approach requires the user to interact with the view (Nodes) to gain additional information on single Nodes or change a Node's value due to the limited space available [NSL18]. In figure 5.4, a small ontology is shown using on–Node mapping.

These representation components do not allow data modification on their own. To allow data modification, we provide a modal containing the Node's data in detail (*Node info modal*). Through navigation, the Node info modal allows for manipulating the values of related nodes. This modal is also available in the treetable view, which we introduce in the following.

Treetable View

The treetable view introduces a representation of the ontology, providing detailed information about each Node to support focus tasks. This view's representation combines a hierarchical tree representation, e.g., used to represent file trees, with a table-based representation of additional Node information. This representation style allows for a quick understanding of each entry's data and interdependencies. The data in columns can be modifiable, showcasing existing interdependencies and the effects of value changes on other Nodes. Table 5.1 showcases a treetable representation of the same small ontology

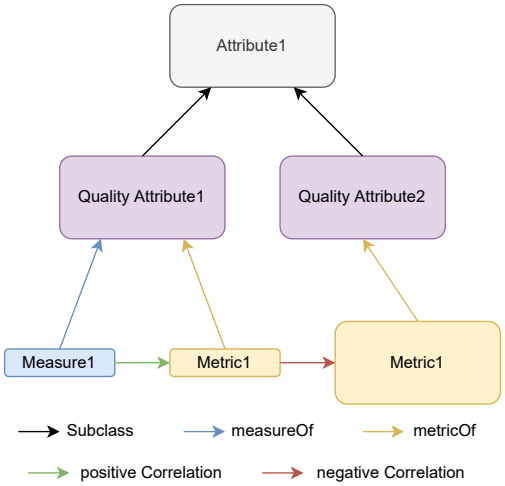


Figure 5.4.: Overview view of an ontology

seen in figure 5.4. The first column includes the hierarchy and Node name, allowing for quick Node identification and ontology traversal. In the second column, the current value is displayed with plus and minus symbols representing the modifiability of the value. The last column includes the positive and negative correlations, with horizontally aligned symbols representing a common correlation. The correlations weight is attached in brackets.

To quickly distinguish the different components of SQOs, we incorporate on-Node mapping, relation, and Correlation coloring. Furthermore, the treetable view allows for future expansion as additional columns can add more data to the view.

However, understanding the ontology's topology is more difficult since only a limited amount of entries can be seen at once, requiring the user to scroll for information about additional entries. The missing topological information results in the treetable view not being adequate for overview tasks.

Node	Value	Correlation
Attribute1 (selected)		
↘ (Subclass) Quality Attribute1		
↘ (measureOf) Measure1	- ■□□□ +	p(1)
↘ (metricOf) Metric1	- ■□□□ +	p(1),n(1)
↘ (Subclass) Quality Attribute2		
↘ (metricOf) Metric2	- ■■■□ +	n(1)

Table 5.1.: Treetable visualization of an ontology

5.4.2. Filters

One key concept of understanding the different aspects of software quality is grasping the effects a change towards one Measure has on other Measures. The selection filter allows users to disable every Node and Arc type, supporting users in focussing their attention on specific, selected components of the SQO. Exemplary usage of the selection filter for focusing on Correlations in a small ontology can be seen in figure 5.5. This figure shows how the selection filter can reduce the displayed relations to not include Subclass relations. The resulting Nodes without any relations are then hidden. Focus can now be laid on the Correlations only.

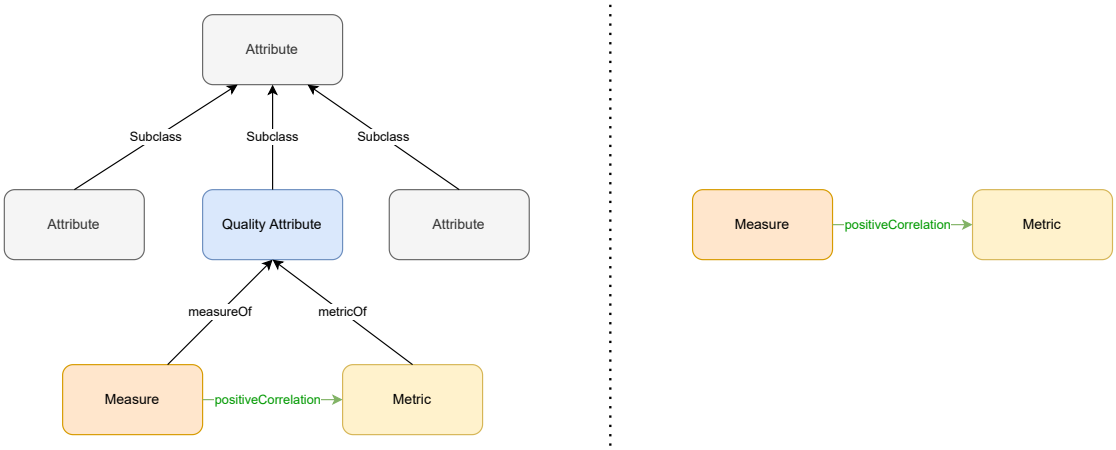


Figure 5.5.: Before and after of selection filter application for focus on Correlations (Subclass relation disabled, not showing isolated Nodes) on overview view

However, some tasks require the visibility of all Node and Arc types around a specific part of the ontology. As numerous Nodes and relations can hinder user focus, an additional depth filter is introduced, limiting each shown Node's distance to the center of attention (the currently selected Node or calculated root Node as described in chapter 5.3). An exemplary ontology with a depth limited to two can be seen in figure 5.6.

This combination of specialized representations for overview and focus tasks with selection and depth filters allows users to create their own ontology chunk representation, selecting only the components related to the current task or goal of understanding.

To extend the support provided by the representation concepts into SQO management, we now introduce our concept for view-supported ontology management.

5.4.3. View-Supported Ontology Management

The views introduced in the previous parts of this chapter incorporate methods to ease understanding through task–focused SQO representation in combination with filters. To further improve the support of ontology management, we introduce a method of Node selection using the represented SQO components in the two views. Currently selected

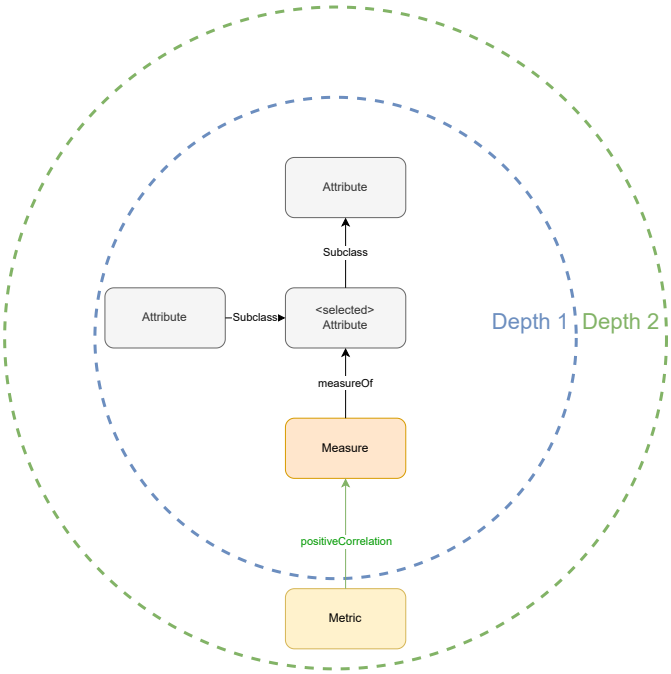


Figure 5.6.: Effect of depth limited to two

Nodes are specially marked using on-Node mapping. Only the task of adding a Node does not use previously represented Nodes. For all other tasks, this concept is applicable and bridges the gap between representation and management.

6. Design

Contents

In order to allow usage of the concepts developed in the previous chapter, a software design is needed. A core component of our design is the replaceability and extendability of the concepts we implement in a visualization.

We begin this chapter with the data storage connection for data persistence. Then, we outline the components of our design model needed for Ontology implementation, with all persisted data included in the models. Additionally, we elaborate on the replaceability and extendability of visualizations incorporating representation concepts. To incorporate new concepts, our work can be reused. Following the ontology class model, we outline the architectural design in the form of our choice to use the MVC pattern. Lastly, we discuss a design decision that, in hindsight, would better suit the modeling of ontologies.

6.1. Data Persistence

Since the research of SQOs is a long-term process, we want SoQOnto to be able to store ontology data for repeated access. Therefore, one crucial aspect of our software design is the *Database* class, bridging the gap between the database server and SoQOnto.

To the *Database* class, we add three types of methods besides the constructor and database connection methods:

1. methods for database management (e.g., `create_db(database: str)`)
2. methods for ontology specific database setup (e.g., `set_graphconfig(database: str)`)
3. methods for ontology management

The data accessed by the Database class is made accessible by our ontology data model, which we introduce in the next section.

6.2. Data Model

We begin this section about our data modeling by introducing the class structure needed to represent ontologies. To enable additional concepts to use this structure, we then describe the design of our extendability and replaceability of visualizations.

6.2.1. Ontology Data Model

To describe the data model of ontologies, we begin with the *Ontology* class containing all of the ontology's data and management methods. An overview of the classes contained can be seen in figure 6.1. In this chapter, we do not describe all classes in detail.

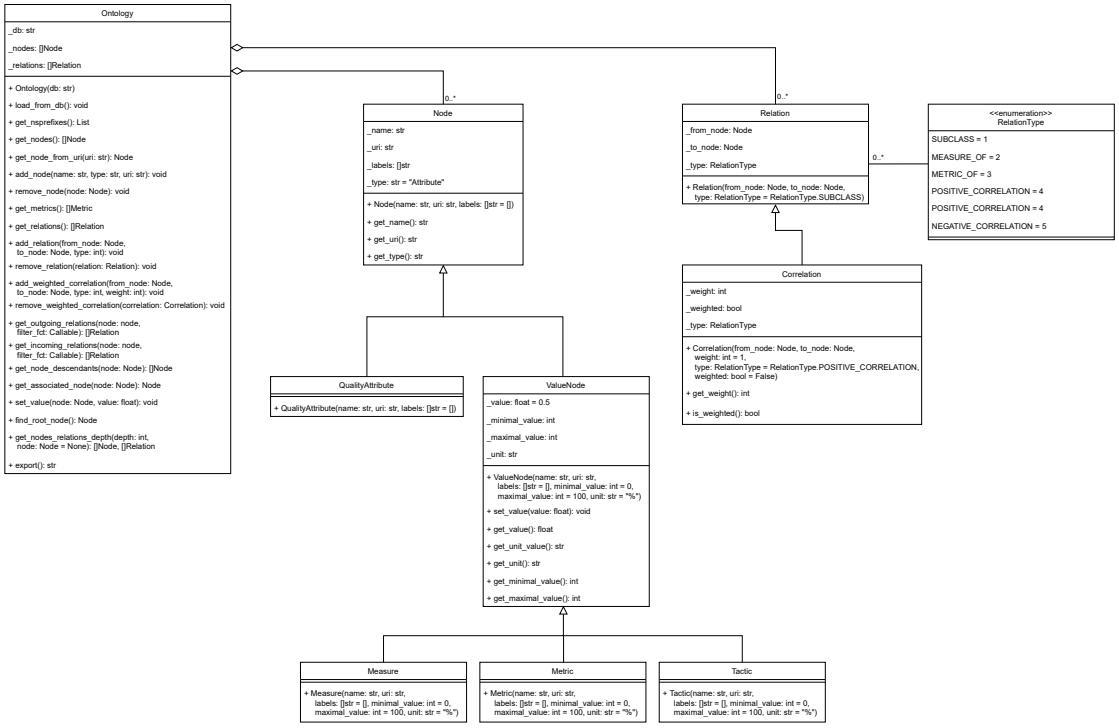


Figure 6.1.: Classes used to represent all parts of an SQO

Ontology Class

The *Ontology* class is the center of our ontology data model. It combines a list of Nodes and relations of different types to an ontology. Therefore, we use this class to implement all ontology management methods.

Node Class

The Ontology class utilizes the *Node* class to store all Nodes contained. Besides its purpose to serve as the most basic Node type (Attribute), it also serves as a starting point for additional Node types, providing essential functionality which can be inherited.

ValueNode Class

To provide a starting point that is inherited by the *Measure*, *Metric*, and *Quality Attribute* classes, we incorporate the *ValueNode* class centrally implementing value–focused methods and attributes. As its sole purpose is to be inherited, we do not instantiate a ValueNode. However, since it inherits from a non-abstract class, it cannot be abstract itself in our development stack.

Relation Class

The *Relation* class stores a relation's two end Nodes as well as the relation's type contained in the *RelationType* enum. The *Correlation* class inherits this class's properties and extends them with a weight used in the value calculation algorithm.

6.2.2. Extendability

To allow the groundwork introduced in the previous sections to be used with additional concepts introduced in the future, which can then be compared to the concepts of this thesis, SoQOnto's implementation of visualizations is focused on extendability and replaceability. We achieve extendability using the *VisualizationManager* class. This class manages the visualization that is shown in the user interface of SoQOnto utilizing the *strategy pattern* [Gam+95]. For all visualizations registered with the VisualizationManager, SoQOnto provides the data source and methods to change components of the SQO represented. To introduce a new visualization, only two methods have to be defined (see figure 6.2). This simple starting point is deliberately chosen so that further functionalities can optionally be implemented. Similar to the *template method pattern* [Gam+95], visualizations can indicate via a variable that a method provided by SoQOnto shall be overridden. The usage of this pattern allows for the fast implementation of new representation concepts. We apply this pattern for the view-supported ontology management concept, of which implementation is optional (see chapter 5.4.3). If the visualization supports view-supported ontology management, it interrupts the ontology management method (e.g., adding a relation) and can resume the process once Nodes are selected.

6.3. Architectural Design

With SoQOnto's data model in place, in this section, we lay our focus on the user interface. For our architectural design, we choose the *Model-View-Controller (MVC)* pattern, allowing for easy maintainability through the separation of concerns between the three

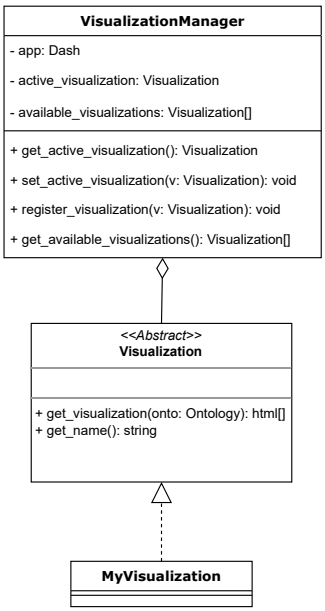


Figure 6.2.: VisualizationManager implementing the strategy pattern

parts. According to Leff et al. [LR01], a hard–to–solve problem is the partitioning of web applications between clients and servers. As we will introduce in the next chapter (See chapter 7.1.1), we develop SoQOnto using a web application framework that combines code for client and server (plotly Dash). Therefore, we do not have to consider this problem. However, a separation of view and controller is only partly possible in this framework. [LR01]

6.4. Discussion

Having introduced this thesis's design components, we discuss a design improvement in this section. When analyzing the components outlined in chapter 6.2.1, to model the different aspects of SQOs, a *composite pattern* [Gam+95] would improve the overall design. However, since we do not utilize this pattern, an anti–pattern in the form of a *God Class* [Del+03] (the Ontology class) is gradually created with each extension of functionality. Therefore, we recommend that future work replace the God Class with a composite pattern, not further increasing the technical debt inflicted.

7. Realization

Contents

To obtain a proof of concept that is usable for and supports research on SQOs, we implement our concepts (chapter 5) and design (chapter 6) in a web–based application we call SoQOnto. Additionally, we utilize the resulting realization in our evaluation in chapter 8. In order to benefit from faster development times and less overhead, we use an agile process for our realization [Han10]. Using this process, we switch back and forth between implementation and conceptualization. This process is especially suitable for us since we create an application based on concepts that are predominantly (on their own or in combination) not verified in practice [BNO13].

This chapter outlines our decision on a development stack, including data storage, then continues with the technical limitations we need to circumvent in our ontology realization. We conclude this chapter with the implementation details of our visualizations incorporating the concepts for human–enabled management.

7.1. Realization Background

7.1.1. Choice of Development Stack

With our concepts and SoQOnto, we want to support the user in efficiently managing SQOs. One prominent focus in the development stack choice is creating an easy–to–access, collaborative tool for research. Therefore, we opt for a web–based rather than a desktop–based application [Jaz07]. When implementing a software tool as a web–based application, increased complexity is commonly introduced through the separation of backend and frontend, written in different programming languages [LR01]. One

web framework bridging this gap between web and desktop–based applications is Plotly Dash[1], introduced by the makers of the widely used plotly.js JavaScript framework.

Plotly Dash combines the popular frontend JavaScript frameworks React.js[2] and plotly.js[3] into a web framework completely accessible in Python. With Plotly Dash, all communication between the *flask*[4] web server and the frontend is abstracted. Therefore, we do not need to implement API endpoints for each communication task, such that we require comparatively little code. By the use of custom React.js components, the development of ontology representation is not restricted, however, introducing a second programming language. We do not implement such custom components in our realization. The incorporation of plotly.js into Plotly Dash is another driving factor for our decision to use Plotly Dash. Plotly.js provides a suite of graph visualization components we can use for ontology representation since ontologies are graphs conceptually.

7.1.2. Choice of Data Storage

As research on software quality is a long–term process, we need data storage to save and provide the data contained in SQOs. Graph–based data storage is a natural choice, as ontologies are, fundamentally speaking, graphs. In this section, we introduce Neo4j as our choice of a graph database and its plugin neosemantics for ontology data storage.

Neo4j

Neo4j[5] is the most popular graph database according to *DB-Engines*' graph database market share analysis, thus being the obvious choice for any development project incorporating the need for storing data naturally in graph form [DB-22]. Since Neo4j does not natively support ontologies' import, export, and modification, its plugin support allows us to extend Neo4j's functionality for our needs.

Neosemantics

Neosemantics[6] is Neo4j's toolkit for support of RDF, and semantics, introducing support for namespaces, mappings, and many more specifically for ontologies. Most importantly, Neosemantics introduces Graph validation, ensuring the correctness of imported and created ontologies natively, removing the need for additional verification of correctness. [neoc]

[1] https://plotly.com/dash/
[2] https://reactjs.org/
[3] https://github.com/plotly/plotly.js/
[4] https://github.com/pallets/flask
[5] https://neo4j.com/
[6] https://neo4j.com/labs/neosemantics/

7.1.3. Deployment

Our choice to develop SoQOnto as a web application should not interfere with the ease of installation. To introduce a controlled environment, we choose docker for deployment[1]. Additionally, with Docker compose[2], the application's installation itself, the database, and its networking are simplified into a single command [RBA17]. With docker's use of a standardized image format [ope], we can also use a kubernetes deployment[3] [Bur+22].

7.2. Ontology Realization

For our ontology realization, we use the Turtle format supported by neosemantics. All vocabulary contained in our ontologies extends the RDFS, OWL, and *qa* ontologies. In this section, we outline our metaclass ontology *qa* defining all components of SQOs, and describe the technical limitations introduced by RDF through its use of triples. We then introduce our solutions to accommodate for these limitations.

7.2.1. Metaclass Ontology (qa)

To use and formalize the components defined in chapter 5.2 in the RDF, we create the *qa Metaclass Ontology*[4], defining the classes present in SQOs. The qa ontology is used by us to define our real–world ontology and represents the ontology format understood by SoQOnto. Figure 7.1 includes all classes and properties found in SQOs. The CanCorrelate class combines all classes that allow for correlations. Outside the "qa" labeled box, used vocabulary from RDF, RDFS, and xsd for data types is shown.

7.2.2. Technical Limitations

In the realization of weighted Correlations, we encountered our only technical limitation in realizing SQOs in the RDF. As discussed in chapter 5.2, some Correlations in SQOs have a more significant impact on the related artifacts' value than others, indicated by a weight. However, turtle definitions consist of triples [Bec14], therefore, not allowing weight to be added to a property, as both end Nodes and the Correlation's type need to be stored as well. These already exhaust the limit of three values that can be stored. In general, one would circumvent this limitation in Turtle by introducing an unnamed artifact (*blank Node*, see chapter 2.3) and storing the additional information needed. See source code 7.1, where the unnamed artifact is contained in lines 5-8. Nevertheless, as neo4j is a graph database, the technical limitation to three values stored per property also applies. Therefore, it cannot store this additional information on the relation itself. To store unnamed artifacts, neo4j generates a unique identifier for the unnamed artifact, allowing for storage in the same way as named artifacts.

[1] https://www.docker.com/
[2] https://docs.docker.com/compose/
[3] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
[4] https://thesis.moritzrickert.de/2022/qa

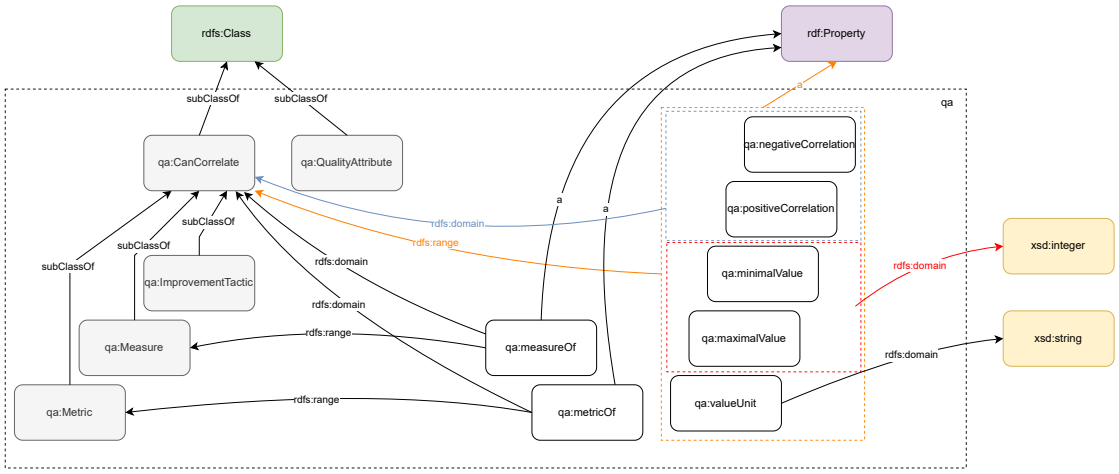


Figure 7.1.: qa ontology class and property definitions not representing solutions to technical limitations. Arrows in the color of a box show a relation of all concepts in the box. “a” relations of classes to rdf:class are omitted.

```
car:RPM
    a qa:Metric ;
    qa:metricOf car:Engine ;
    qa:correlated [
        a qa:Correlation ;
        qa:correlationTo car:Speed ;
        qa:correlationType "positive" ;
        qa:correlationWeight 3 ;
    ] ;
    rdfs:label "RPM" .
```

Source Code 7.1: Circumventing the triple’s limitation with an unnamed artifact

When exporting the SQO in Turtle format, neo4j now cannot differentiate between previously named and unnamed artifacts. Therefore, it does not add the unnamed Correlation artifact to the corresponding artifact (source code 7.1) but as a new artifact with its generated unique identifier (see source code 7.2). The resulting exported file differs content-wise from the imported file, making use of an ontology exported from SoQOnto in other tools cumbersome.

```
car:RPM
    a qa:Metric ;
    qa:metricOf car:Engine ;
    qa:correlated b_25876 ;
    rdfs:label "RPM" .

<bnode://node1gctlkt0nx3>
    a qa:Correlation ;
    qa:correlationTo car:Speed ;
    qa:correlationType "positive" ;
    qa:correlationWeight 3 .
```

Source Code 7.2: Problem when exporting ontology containing unnamed artifacts from neo4j

We opt for named, weighted Correlations to circumvent a renaming of artifacts and prevent content-wise changes on export. This solution provides both the ability to set a Correlation's weight and export with unchanged artifacts (source code 7.3), still providing a basic definition of Correlations (*qa:positiveCorrelation*, *qa:negativeCorrelation*) initialized in SoQOnto with a weight of 1. Since this differentiation is only based on the limitation of the RDF, we treat both correlation types the same.

```
car:RPM
    a qa:Metric ;
    qa:metricOf car:Engine ;
    rdfs:label "RPM" .

car:RPMSpeedCorrelation
    a qa:Correlation ;
    qa:correlationType "positive" ;
    qa:correlationWeight 3 ;
    qa:correlationFrom car:RPM ;
    qa:correlationTo car:Speed .
```

Source Code 7.3: Adding a weighted Correlation with a named Correlation artifact

7.3. Base Application Realization

In this section, we outline the realization of the functionality SoQOnto provides to each visualization (the base application). Besides the ontology modification methods, which are interceptable by visualizations as described in chapter 6.2.2, the export and import of ontologies are handled using the Turtle file format. In this section, we outline the exchange of data between the base application (SoQOnto) and visualization. Then, we describe how visualizations indicate their support for optional functionality, e.g., view–supported ontology management (chapter 5.4.3).

7.3.1. Exchanging Data with Visualizations

When implementing a visualization, retrieving data from the base application is easy. The base applications data structure is previously defined and known at the implementation and design stages. However, providing data to the base application is a more complex issue, as the base application cannot and should not have to be modified to suit every visualization.

To achieve data exchange from the visualization to the base application, we use *pattern–matching callbacks*, which listen to changes of HTML elements matching a given pattern. For example, to open the new relation modal from the visualization, it only has to register a dcc.Store element[1] for data storage with an id in the form of `{"open_new_relation_modal": "my_visualization_p1"}`. The pattern–matching callback will then open the modal using the data provided. To prevent the base application from opening the modal before the data (e.g., end Nodes) is available, the visualization must indicate the implementation of optional functionality. We explain this process in the next section.

7.3.2. Optional Visualization Functionality

To achieve a low bar for the implementation of new visualization concepts, a simple, working visualization in SoQOnto does not require the implementation of many methods. However, if a visualization implements more functionality than required, this has to be indicated.

```
    def __init__(self) -> None:
    """
    __init__ extends 'Visualization' constructor with optional functionality
    """
    super().__init__()
    self._name = "My Visualization"
    self._supports_custom = {
        'add-relation': True,
        'remove-relation': True,
        'remove-node': True
    }
    self._supports_ontology_change_callbacks = True

```

Source Code 7.4: Constructor for a visualization with all optional functionality

To only reload visualization parts affected by ontology changes, the parameter *__supports_ontology_change_callbacks* defines whether the visualization listens for ontology changes. Otherwise, SoQOnto reloads the visualization to adapt it to the changes. Additionally, a visualization can implement view-supported ontology management, indicated through the *__supports_custom* set. If a visualization indicates usage of, e.g., a custom end Node selection for new relations (the “add-relation” key), standard end Node selec-

[1] https://dash.plotly.com/dash-core-components/store

tion is canceled, and the visualization must provide the end Nodes selected by the user. An exemplary indication of the implementation of all optional functionality is shown in source code 7.4.

7.4. Basic Visualization

As a reference for evaluating our concepts in chapter 8, we implement a basic visualization, providing only a graph of the ontology with on–Node mapping of both the names and types of Nodes and relations as seen in figure 7.2. This visualization does not provide any optional visualization functionality and does not claim to be better suited for the management of SQOs than other existing tools. Rather, it represents a minimal, running visualization prototype in SoQOnto. Through comparison with the Advanced Visualization, containing all concepts of chapter 5, introduced in the next section, we evaluate the concepts' support in human–enabled management of SQOs.

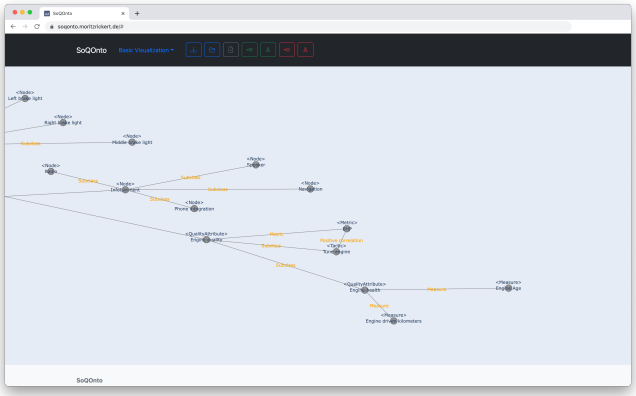
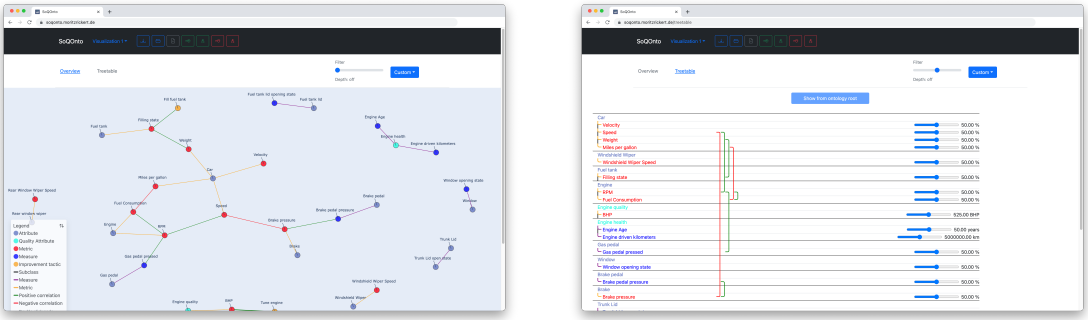


Figure 7.2.: Parts of the real–world ontology are visualized using the basic visualization

7.5. Advanced Visualization

In contrast to the Basic Visualization, the Advanced Visualization implements all concepts and optional functionality introduced in chapter 5. For task–focussed ontology representation, it provides the overview and treetable views, complemented with filters. A comparison of the two views, filtered for focus on correlations through the hiding of subclass relations and isolated nodes, is shown in figure 7.3. The figure also displays the concept of on–Node mapping of component types through the coloring of Nodes and relations. It can be seen that the modification of a Node's value is a central part of the treetable view for focus tasks, as exact values and an input for modification are visible at all times. In contrast, the overview view focuses on the topology of the represented artifacts. On ontology change, the visualization reloads only the affected component

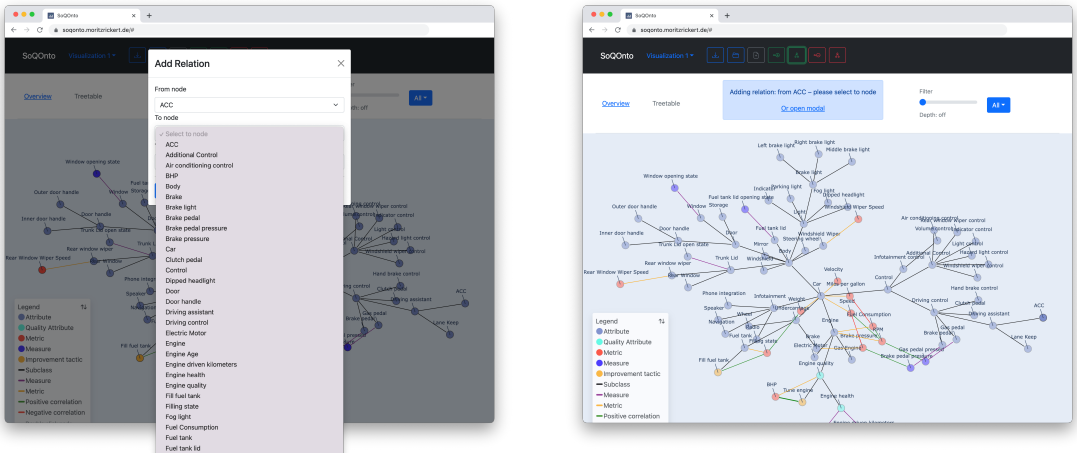
(e.g., the graph in the overview view), such that the set filters are not reset by reloading the page itself.



(a) Overview view (b) Treetable view

Figure 7.3.: Overview and treetable view of the Advanced Visualization

The Advanced Visualization additionally implements the concept of view–supported ontology management. To select end nodes, e.g., for adding a new relation, nodes in the graph or tree can be selected by clicking them. Figure 7.4 shows a comparison of the complexity of node selection using the view–supported method and base application method. With the view–supported method, modifications are possible while staying focused on the view.



(a) Base application Node selection mechanism

(b) View–supported Node selection mechanism in the Advanced Visualization's overview view

Figure 7.4.: A comparison of view–supported Node selection vs. an html select element

8. Evaluation

Contents

To prove the support of our concepts in the human–enabled management of SQOs, we conduct a quantitative evaluation in the form of a user survey. We choose a user survey over interviews, as user surveys allow for asynchronous participation and better comparison of the answers given [Sch13]. In the user survey, we compare the Basic Visualization with our Advanced Visualization, incorporating all concepts outlined in chapter 5, and have users rate both the usability of the system and the visualization's support in the given tasks. Subsequently, we discuss the results of the evaluation. We then answer our research questions previously outlined in chapter 3. Lastly, we consider the limitations of our research.

8.1. Quantitative Evaluation – User Survey

In this section, we describe the setup of our evaluation, including a case–study ontology of the field of software quality, the tasks, and the questionnaire. Then, we outline our analysis method, followed by the user survey's results, and conclude with a discussion of the results. In section 8.1.3, we present the answers to our evaluation questionnaire. We discuss the presented results in the following section.

8.1.1. Setup

To enable participants to complete our user survey on their own time, we construct it as an at-home, non-live user survey. All resources needed for the completion of the evaluation are provided by us online: (1) a short introductory presentation in pdf and video format outlining SoQOnto and the visualizations, (2) a task sheet (Appendix A) describes the tasks we ask the participants to complete with (3) SoQOnto on a given case–study ontology, and (4) an evaluation questionnaire. To exclude the factor of order, we use a crossover AB/BA design [MK18]. Hence, half of the participants start

with the Basic Visualization (Task sheet A). The other half starts with the Advanced Visualization (Task sheet B).

In the following, we introduce a case–study ontology on the field of software quality, which we then use to evaluate our concepts.

Case–Study Software Quality Ontology

To evaluate our concepts for human–enabled management of SQOs, we provide an ontology incorporating all components described in chapter 5.2. Our real–world ontology incorporates these components and removes the abstractness and complexity of software quality. However, it differs structurally from ontologies on the domain of quality. Therefore, for the evaluation, we create a case–study ontology on the field of software quality based on the SQuaRE software quality model and additional research conducted at our chair for software construction, modeling correlations of the Quality Attributes Maintainability and Portability in Microservice Architectures [iso]. We do not claim complete coverage of software quality in this ontology. Instead, we aim to implement all core components while not going beyond this thesis's focus and allowing the participants of our evaluation to gain an overview in the limited time available to them.

We center our SQO around the SQuaRE software quality model, incorporating all Quality Attributes and Factors for a hierarchical core ontology with 40 nodes [iso]. As SQuaRE only contains taxonomical aspects, to obtain a SQO incorporating all components, we extend this groundwork with Correlations and Metrics substantiated in a systematic literature review at our chair. The resulting case–study ontology is shown in figure 8.1, consisting of the eight Quality Attributes described in SQuaRE, subdivided into their respective Quality Factors. For the Quality Attributes portability and maintainability, we include Measures and Metrics. Metrics that are calculated using multiplication and division are approximated using addition and subtraction where possible.

Tasks and Questionnaire

The participants are asked to complete the following tasks to evaluate our concepts for support in the modification and representation of SQOs:

Ontology management: to evaluate view–supported ontology management and the import method of SoQOnto, the participants have to import the case–study ontology, remove, and add a Node and multiple relations.

Understanding of correlations: to evaluate the visualization's support in understanding the effects of correlations, the participants are asked to find all Nodes increasing the Metric "Effort of Package Construction."

After completing the tasks, the participants fill out an online questionnaire on a LimeSurvey instance[1]. The questionnaire starts with demographic questions about gender, age, and previous knowledge about graphs and software quality (attributes).

[1] https://www.limesurvey.org/

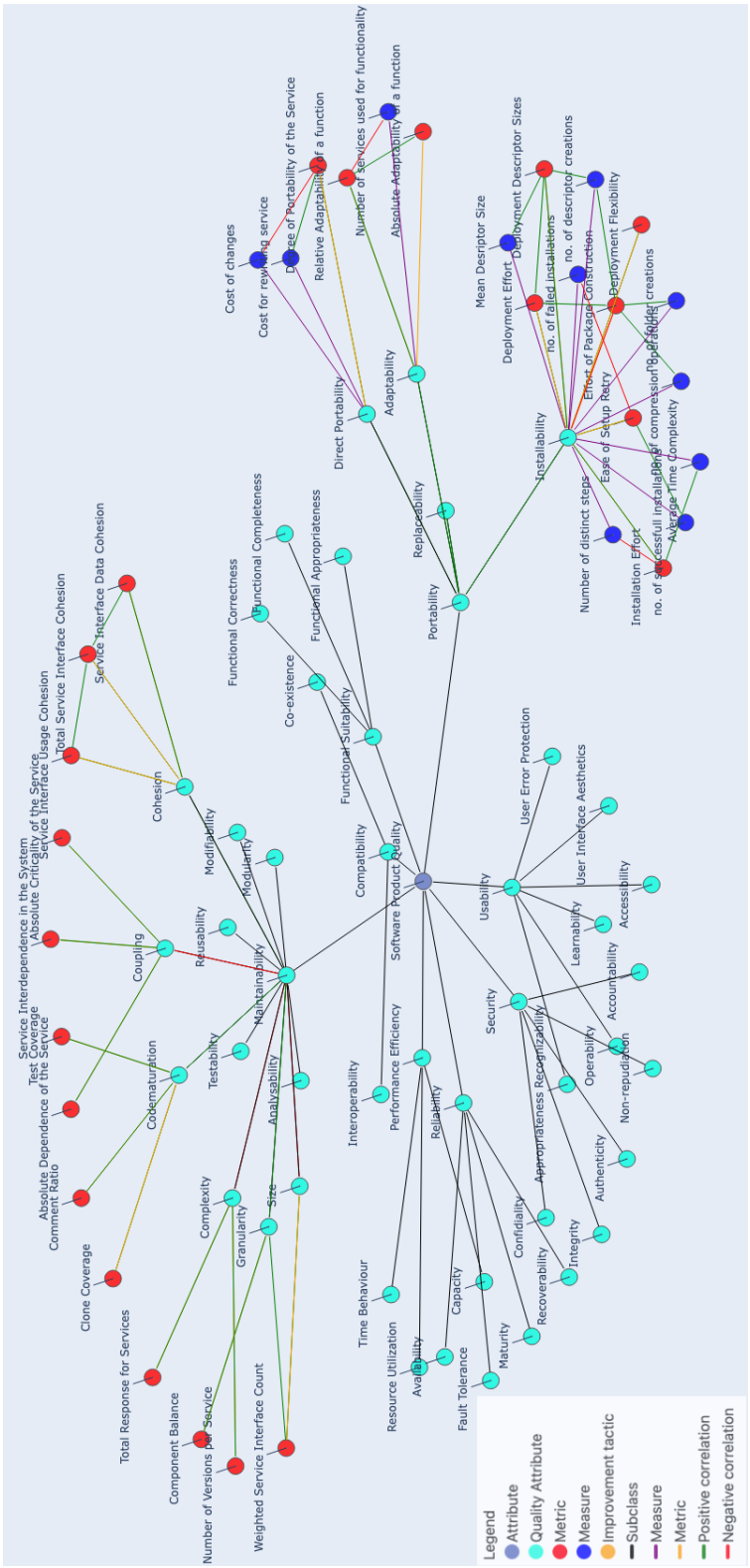


Figure 8.1.: Case–study ontology based on SQuaRE, extended with research conducted at our chair [iso]

The following questionnaire section is based on the widely used *System Usability Scale (SUS)* [Bro95], consisting of 10 questions. The SUS offers an easy–to–use questionnaire, rigorously validated and widely adopted for evaluation of user experience [BKM08]. It measures the three essential aspects of system usability: system effectiveness, system efficiency, and system satisfaction [18]. We extend the SUS with four rating questions targeting towards the visualization's support in task completion and three free text questions to gain insights into positive and negative user experiences as well as possible improvements. All rating questions are rated on a 10–point Likert scale from "strongly disagree" to "strongly agree." The complete questionnaire can be found in Appendix B. To provide help with questions and problems encountered by the participants, we schedule two video meetings.

8.1.2. Analysis

According to the study design of the evaluation, in the following sections, we describe the results of our quantitative evaluation both textually and visually using figures. We start by checking the demographics of our participants for bias. Then, we check the differences in results for statistical significance using Student's t–tests for connected samples. If the data is not normally distributed, we use the Wilcoxon signed–rank test to test for statistical significance. The normality of the distribution of our results is determined using the Shapiro–Wilk normality test. For all statistical tests, we use a confidence level of 95 % ($\alpha = .05$).

8.1.3. Results

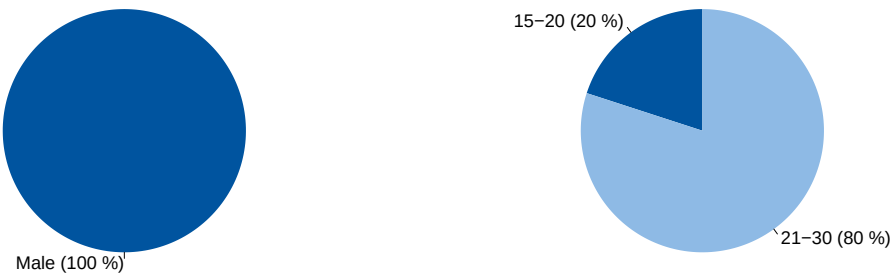
In this section, we introduce the quantitative evaluation's results. The total number of participants is 15, with 7 participants using task sheet A and 8 using task sheet B. We begin by outlining the demographics of the participating students, then inspecting the usability. We then continue with the additional questions, answered on a scale, too, concerning the support in task completion, and conclude with free text answers to questions concerning the visualizations.

Demographics

All participants identify themselves as male (see Figure 8.2a). Most participants are between 21 and 30 years old, with 20 % younger at ages 15 to 20 (see Figure 8.2b).

We conclude the demographics question group with the participants' familiarity with graphs and software quality. The answers to these three statements are visualized in Figure 8.3.

The first statement, "I am familiar with graphs," is rated by the participants with an average of 7.87 ($SD = 2.17$), transferring to *agree*, as does the median of 8. The minimum is *strongly disagree*, and the maximum *strongly agree*. The answers' first quartile corresponds to *agree*, and the third quartile is between *agree* and *strongly agree*.



(a) Gender (b) Age

Figure 8.2.: Gender and age of participants ($N = 15$)

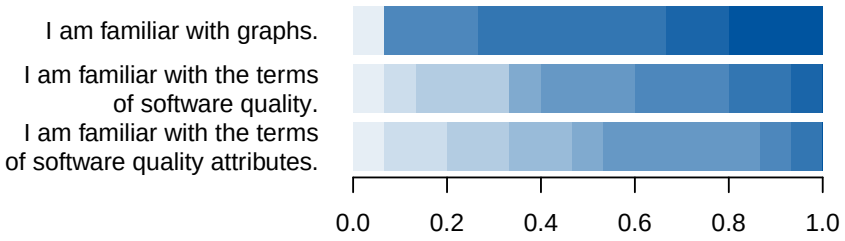


Figure 8.3.: Familiarity of participants with graphs and software quality (attributes), 10-point Likert scale, darker is better ($N = 15$)

The average of the second statement, "I am familiar with the terms of software quality," is 5.4 ($SD = 2.44$) with a median of 6 (both *neutral*). The participants answered with a minimum of *strongly disagree* and a maximum between *agree* and *strongly agree*, with the first quartile of *disagree* and the third quartile between *neutral* and *agree*.

The third statement's answers ("I am familiar with the terms of software quality attributes") average at 4.6 ($SD = 2.06$) between *disagree* and *neutral*, and a median of *neutral*. The minimum is *strongly disagree*, and the maximum *agree*. The first quartile corresponds to *disagree*, and the third quartile to *neutral*.

System Usability Scale

For easy comparison, the questions in the SUS are grouped by questions rather than visualizations. A visual comparison can be seen in Figure 8.5.

In the SUS score rescaled to a minimum of 0 and a maximum of 100, the Basic Visualization achieves a score of 49.8, significantly lower than the Advanced Visualization's

score of 78.9. According to Bangor et al., this results in the Advanced Visualization's score being above average for Web-based interfaces (68.05) and the Basic Visualization being below average [BKM08]. In further research, Bangor et al. couple adjectives with SUS scores [BKM09]. Using this connection, we can describe the Advanced Visualization as between *good* and *excellent*, while the Basic Visualization is best described as *ok*.

Without any exception, the average score of each SUS question is higher for the Advanced Visualization than for the Basic Visualization (See Figure 8.5). The lowest difference in average score is found for the questions 6 "I thought there was too much inconsistency in this system" (Difference: 1.0, see Figure 8.5f), 4 "I think that I would need the support of a technical person to be able to use this system." (Difference: 1.3, see Figure 8.5d), and 10 "I needed to learn a lot of things before I could get going with this system" (Difference: 1.4, see Figure 8.5j). All other response averages differed by at least 2.2.

To determine the statistical significance of the differences described, we calculate the SUS for each participant and check the resulting data set for normality. The Shapiro–Wilk normality test tests our null hypothesis that the data set is normally distributed. It returns a p-value of .199. Therefore, our data set can be assumed to be normally distributed, allowing us to use Student's t–test. Using Student's t–test, we receive a p-value of .0003, below our confidence level of .05. Therefore, the difference between the Basic and Advanced Visualization is *statistically significant*.

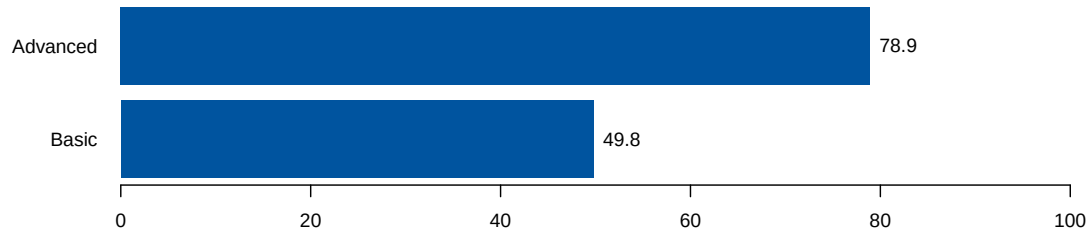
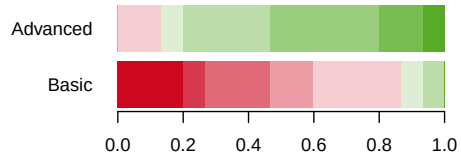


Figure 8.4.: Total System Usability Scale score (normalized to 100) ($N = 15$)

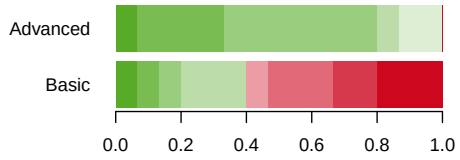
Further Questions on the Understanding of Software Quality and Task Support

As with the SUS questions, the additional questions on the understanding of software quality and task support of the visualizations are rated higher for the Advanced Visualization than for the Basic Visualization (See Figure 8.6). For these questions, the lowest difference in average scores is 2.5 (See Figure 8.6a), and the highest is 4.1 (See Figure 8.6d).

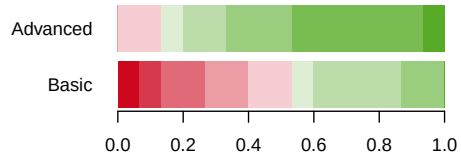
In addition to the already described questions rated on a scale, we ask the participants which aspects of each visualization they like, do not like, and what could be improved. A complete list of all answers can be found in Appendix C.



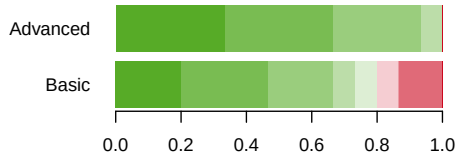
(a) "I think that I would like to use this system frequently."



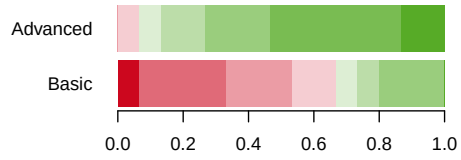
(b) "I found the system unnecessarily complex."



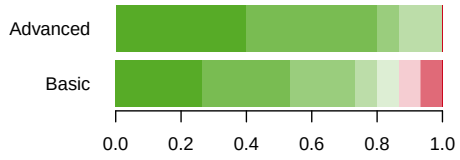
(c) "I thought the system was easy to use."



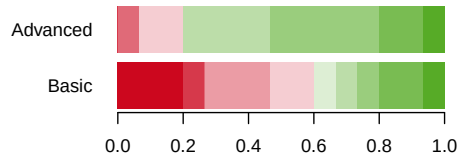
(d) "I think that I would need the support of a technical person to be able to use this system."



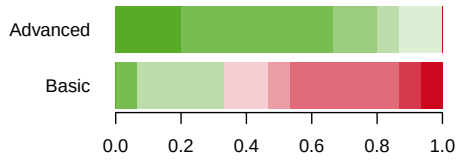
(e) "I found the various functions in this system were well integrated."



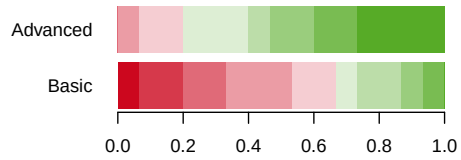
(f) "I thought there was too much inconsistency in this system."



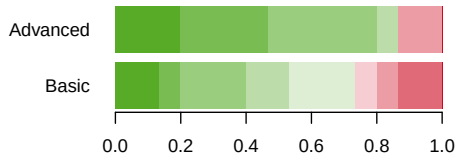
(g) "I would imagine that most people would learn to use this system very quickly."



(h) "I found the system very cumbersome to use."



(i) "I felt very confident using the system."

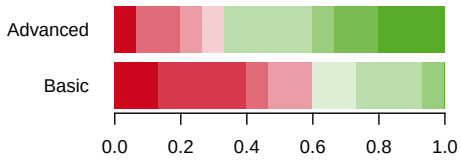


(j) "I needed to learn a lot of things before I could get going with this system."

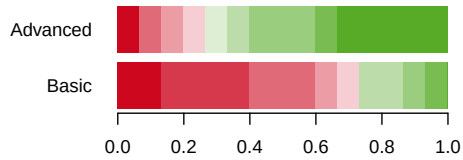
Figure 8.5.: System Usability Scale ratings comparison by question, 10-point Likert scale, from *strongly disagree* (red) to *strongly agree* (green)

For the Basic Visualization, positively described aspects included 1. simple Node selection, 2. explicitly stated Node types, and 3. representation as a graph. The participants do not like 1. too many (overlapping) labels, 2. lack of color, and 3. text color. As described by the participants, improvements could be 1. spreading the graph more, 2. adding color and removing some text, and 3. a search function.

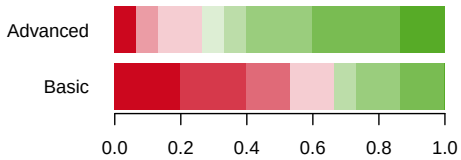
The participants' positive feedback on the Advanced Visualization includes 1. colored Nodes, 2. filters, and 3. the legend. They do not like 1. the restriction of the Node selection, 2. the unintuitive zoom function, and 3. the hardness of finding a specific Node. Therefore, the proposed improvements include 1. a search function, 2. a combination of both visualizations' Node selection, and 3. better graph interaction.



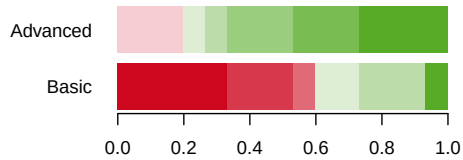
(a) "I feel that my understanding of software quality and its interdependencies increased by using this visualization."



(b) "I feel that the visualization elements of this visualization supported me in my tasks."



(c) "I feel that the visualization elements of this visualization supported me in my understanding of software quality."



(d) "I feel that the visualization elements of this visualization work well together and complement each other."

Figure 8.6.: Additional questions ratings comparison by question, 10-point Likert scale, from left (*strongly disagree*, red) to right (*strongly agree*, green)

8.1.4. Discussion

It has to be mentioned that the gender composition of the participants is not representative of the general population. Additionally, our participants are all younger than 30 years old and already have high knowledge of graphs and some knowledge about software quality. While this might not be the case for the general population, as researchers and other interested parties, consisting of developers and product owners, primarily conduct SQO management, some knowledge about graphs and software quality should be assumed.

The stark difference in scores achieved by the Basic and Advanced Visualization in this

evaluation underlines the effectiveness of the concepts introduced in this thesis and used in the Advanced Visualization. Additionally, as a surplus, it must be noted that SoQOnto allows for ontology management with little to no help from a technical person, as shown in the good scores for both visualizations in the related questions. This supports our impression while conducting the evaluation, as only one participant is using the support video meetings.

8.2. Answers to the Research Questions

In this section, we present the contribution of our work to the research questions we extracted from our problem statement in chapter 3.

> **RQ1:** *How can a real–world ontology, incorporating all relevant aspects of a Software Quality Ontology, look like?*

Following the introduction of our real–world ontology in chapter 3.1.1, we conclude that our car ontology covers all aspects of SQOs. Through the familiarity of the car domain, it is easy to understand and allows for focus on the representation concepts. However, to create a structurally more similar ontology to SQOs, we suggest to narrow the domain towards car *quality* while preserving this domain's less abstract and more familiar nature.

> **RQ2:** *How can a visual representation support the understanding of the components and relations of SQOs?*
>
> > **RQ2.1:** *How can the effects of Correlations between Measures and Metrics be visualized?*
> >
> > **RQ2.2:** *How can the aspects of quality be represented in an understandable way?*

The answer to RQ2 consists of the answers to both RQ2.1 and RQ2.2.

To visualize the effects of Correlations between Measures and Metrics, we utilize on–Node mapping with differing Node sizes and layout adaption. For the visualization of Node values, we need to compute values taking weight and correlations into account.

To represent the aspects of quality in an understandable way, we differentiate between two classes of ontology tasks and incorporate both a graph–based overview view using on–Node mapping and a treetable view combining the hierarchical structure of a tree with a table for additional Node data.

Combined, these concepts increase the participants understanding of software quality and provide a statistically significantly better user experience of 78.9 versus 49.8 in the SUS score.

> **RQ3:** *How can the modification and sharing of a represented ontology be simplified?*

For ontology modification, we integrate the concept of view–supported ontology management, allowing for Node selection using the visualization's representation of Nodes. We are able to show that the textual RDF syntax of Turtle is suitable for the export and import of SQOs in SoQOnto, therefore, allowing for sharing of knowledge. Participants of the evaluation positively mentioned the intuitiveness of view–supported ontology management. They do not feel the necessity to consult a technical person when using SoQOnto and its Advanced Visualization.

8.3. Limitations

While the overall development and evaluation of our visualizations and SoQOnto confirms the concept, design, and realization, some limitations came to light.

Dependence on Graph Visualization in the Advanced Visualization

A commonly raised topic in the free text part of the evaluation is that users would like to select the end Nodes from a dropdown – like in the Basic Visualization – or at least have the option to do so, stating the difficulty of finding Nodes in large ontologies. In the current version of the Advanced Visualization, it is the user's choice to select end Nodes in the graph representation or the dropdown.

Correlations do not Support Multiplication or Division

When creating the case–study ontology for the field of software product quality, some Metrics introduced in the research used are a product or quotient. Due to the current value calculation algorithm (Chapter 5.3.1), allowing for only addition and subtraction, we can only approximate the value of these Metrics.

9. Conclusion and Future Work

Contents

9.1. Conclusion

This thesis aims at enhancing the understanding of software quality with the help of well–known and new concepts for the representation of SQOs. To provide researchers with a supportive environment for understanding and maintaining SQOs, we focus on the representation of fundamental components and the effects of Correlations between the different aspects of software quality. In the following, we summarize our findings:

- To reduce abstractness and complexity found in software quality and increase familiarity, we define a real–world ontology on the field of a car as an everyday item. The usage of our real–world ontology allows for a focus on the representation and visualization concepts. We learn that for a structurally more similar ontology to SQOs, our real–world ontology needs a narrowing of domain towards a car *quality* ontology.

- To achieve a goal–oriented representation of the components of SQOs, we incorporate an ontology task classification, allowing us to complement the different requirements of both *overview* and *focus tasks*. We represent these task classes in our visualization through an *overview* and *treetable view*, combining a tree structure with a table for the representation of additional Node data. In both views, we use on–Node mapping to represent the Node data, including the artifact's type (Measure, Metric, ...).

- In a third step, we focus on the representation of the effects of Correlations. To represent the Correlations found in software quality, we calculate the values of correlated Nodes on value change. We represent the calculated value in the overview view through on–Node mapping, increasing or decreasing the Node's size. In the treetable view, as we represent the Node's definite value at all times, its value is updated.

- With our concepts for the representation of SQOs introduced, we then focus on maintaining SQOs. To achieve effective maintainability, we introduce view–supported ontology management, utilizing the SQO's representation for maintenance.
- Finally, we evaluate our concepts for representing and maintaining SQOs through comparison with a minimal, running visualization in a user survey. We implement both visualizations in SoQOnto, a usable proof–of–concept web application we provide in the evaluation. Using the widely adopted *System Usability Scale (SUS)*, we conclude that the increase in usability for both the maintenance and representation concepts is statistically significant. Extending the SUS, we find that the participants' understanding of software quality increases more with than without the application of our concepts. Nevertheless, we find that in practice, the computation of a Node's value sometimes requires multiplication or division, which our value calculation algorithm does not support so far.

Overall, our concepts contribute to both the understanding and maintenance of SQOs. Together, the visualization, including our concepts, and their realization in SoQOnto provide good to excellent usability.

9.2. Future Work

While implementing SoQOnto, the visualizations, and ontologies, we found opportunities to improve the versatility of SoQOnto and the SQO metaclass ontology (*qa*). Additionally, the feedback of the participants of our user survey indicates possible improvements for the Advanced Visualization. Concluding this thesis, we want to outline possible improvements that can, in our view, enhance our web application and concepts.

Allowing for Different SQO Metaclasses: In the current implementation of SoQOnto, the names of metaclasses to be expected from any ontology imported are pre–defined. To allow for more versatility, it might be useful to allow users to specify Node and relation classes corresponding to the concepts contained in SQOs. An implementation of this kind would allow ontologies with similar characteristics that are not defined using the *qa* metaclass ontology to profit from the methods and representations included in SoQOnto's visualizations.

Implementing Additional Correlation Types and Custom Value Composition: The *qa* metaclass ontology's vocabulary incorporates two Correlation types (positive and negative Correlations). Currently, the sum of their (adjusted) values comprises the value of a correlated Node. In the late stages of this thesis work, we implement an ontology for software quality, of which some Metrics could not be precisely represented. As these Metrics' values are calculated using multiplication and division, positive and negative Correlations are not suitable for precise representation. Further research on different compositions of a Metric's value and correspondingly into a value calculation algorithm that incorporates the found composition types is, in our view, justified.

A. Evaluation Tasks

See next pages for evaluation task sheet. The task sheet of group B only differs in the order of tasks 1 and 2.

Evaluation Group A: Human–enabled management of software quality ontologies

Moritz Rickert
moritz.rickert@rwth-aachen.de

September 5, 2022

Thank you for taking your time to participate in the evaluation of my thesis titled "Human–enabled management of software quality ontologies."

The evaluation consists of two parts: a series of basic tasks to be conducted on both the basic and advanced visualization of the tool and a survey about your experience prior to using the tool as well as after your usage.

If you have any questions regarding the tool or evaluation, feel free to participate in one of the zoom meetings or simply write me an email. I'll be happy to answer your questions.

Zoom meetings
Thursday, September 8, 2022 18:00 – 19:00
Sunday, September 11, 2022 18:00 – 19:00

1 Basic visualization

You can find the basic visualization of SoQOnto at **https://soqonto-basic.moritzrickert.de**. Please take a minute to make yourself comfortable with the user interface.

1.1 Import an ontology

Click on the "import ontology" button and follow the steps in the modal to import the ontology found at **https://thesis.moritzrickert.de/2022/eval**.

Note: Please use a name of your choosing (e.g. your initials etc.) as the ontology's name has to be unique.

1.2 Gain an overview

You are now presented with a graph representation of the ontology. Take a moment to gain a broad understanding of the ontology.

Note: You do not have to understand the ontology in detail.

1.3 Manage the ontology

The Quality Attribute "Replaceability" (here: "Replacability") has a spelling mistake and is connected to the Quality Attribute "Compatibility". However, it should be connected to the Quality Attribute "Portability".

- Remove the Quality Attribute "Replacability"
- Again add the Quality Attribute with its correct name
- Connect the Quality Attribute as a subclass to "Portability"
- Add a positive correlation from "Replaceability" to "Portability"

1.4 Effects of value changes

Find the Metric "Effort of Package Construction". Try to understand whether an increase or decrease of the value of this Metric is needed to increase software quality. Which values would have to increase in order to increase the Metric "Effort of Package Construction".

Note: a positive correlation implies that an increase in value of one connected node leads to an increase in value of the other. A negative correlation implies that an increase of value of one node leads to a decrease of value of the other.

2 Advanced visualization

You can find the advanced visualization of SoQOnto at **https://soqonto-advanced.moritzrickert.de**. Please take a second to make yourself comfortable with the user interface. You are free to use either view contained in the advanced visualization to complete the tasks.

2.1 Import an ontology

Click on the "import ontology" button and follow the instructions to import the ontology found at **https://thesis.moritzrickert.de/2022/eval**.

Note: Please use a name of your choosing (e.g. your initials etc.) as the ontology's name has to be unique.

2.2 Gain an overview

You are now presented with a graph representation of the ontology. Take your time to understand the ontology's aspects.

2.3 Manage the ontology

The Quality Attribute "Replaceability" (here: "Replacability") has a spelling mistake and is connected to the Quality Attribute "Compatibility". However, it should be connected to the Quality Attribute "Portability".

- Remove the Quality Attribute "Replacability"
- Again add the Quality Attribute with its correct name
- Connect the Quality Attribute as a subclass to "Portability"
- Add a positive correlation from "Replaceability" to "Portability"

2.4 Effects of value changes

Find the Metric "Effort of Package Construction". Try to understand whether an increase or decrease of the value of this Metric is needed to increase software quality. Which values would have to increase in order to increase the Metric "Effort of Package Construction".

Note: a positive correlation implies that an increase in value of one connected node leads to an increase in value of the other. A negative correlation implies that an increase of value of one node leads to a decrease of value of the other.

3 Survey

Please open the survey found at **https://survey.moritzrickert.de/index.php/642519** and answer all given questions.

Thank you again for your participation.

B. Evaluation Questionnaire

All questions are answered in a range of 1–10 if not stated otherwise.

B.1. Demographics

1. Gender (male, female, diverse)
2. Age (15-20, 21-30, 31-40, 41-50, 51-60, ...)
3. I am familiar with Graphs.
4. I am familiar with the terms of software quality.
5. I am familiar with the terms of software quality attributes.

B.2. Basic Visualization

B.2.1. System Usability Survey

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

B.2.2. Additional Questions

1. I feel that my understanding of software quality and its interdependencies increased by using this visualization.

2. I feel that the visualization elements of this visualization supported me in my tasks.

3. I feel that the visualization elements of this visualization supported me in my understanding of software quality.

4. I feel that the visualization elements of this visualization work well together and complement each other.

5. What did you like about the basic visualization? (free text)

6. What did you not like about the basic visualization? (free text)

7. What could be improved with the basic visualization? (free text)

B.3. Advanced Visualization

B.3.1. System Usability Survey

1. I think that I would like to use this system frequently.

2. I found the system unnecessarily complex.

3. I thought the system was easy to use.

4. I think that I would need the support of a technical person to be able to use this system.

5. I found the various functions in this system were well integrated.

6. I thought there was too much inconsistency in this system.

7. I would imagine that most people would learn to use this system very quickly.

8. I found the system very cumbersome to use.

9. I felt very confident using the system.

10. I needed to learn a lot of things before I could get going with this system.

B.3.2. Additional Questions

1. I feel that my understanding of software quality and its interdependencies increased by using this visualization.
2. I feel that the visualization elements of this visualization supported me in my tasks.
3. I feel that the visualization elements of this visualization supported me in my understanding of software quality.
4. I feel that the visualization elements of this visualization work well together and complement each other.
5. What did you like about the advanced visualization? (free text)
6. What did you not like about the advanced visualization? (free text)
7. What could be improved with the advanced visualization? (free text)

B.4. Additional Feedback

Free text answers.

C. Evaluation Feedback

Question: What did you like about the basic visualization?

- “The navigation elements e.g. the Zoom”
- “Graph drawing is always nice to locate clusters etc. and transitive relationships”
- “Simple input methods”
- “It is very good to see the different abstract software quality aspects in a graphic way”
- “It’s simplicity and that it is very easy to add/delete nodes/relations ”
- “Import interface and flow”
- “See what type a Node has”
- “-”
- “The color scheme was nice to look at.”
- “fast overview, the type of the nodes is explicitly stated(metric, measure...)”
- “The function of zooming in made it clearer. Editing was user-friendly.”
- “easy to understand”, “Intuitive operation”
- “The ability to add relations to nodes by entering their names and not searching their visual representation in the graph”
- “that you only worked with the buttons on the top side and no double klicking on the nodes”
- “No disadvantage for color-blind people.”

Question: What did you not like about the basic visualization?

- “”Descriptors” like <Metric> lead to an unnecessary complication of the graph making it hard to find certain elements.”
- “Overlapping labels, overall graph visibility due to label size not adapting to proximity of other nodes, makes locating nodes very tricky”

- "Hard to comprehend graph on small screen (14")"
- "There are to many labels that make it difficult to see through. It is hard to diferenciate between different software aspects. It is hard to find the root node and subclasses are not marked different than their parents. You cant see correlations"
- "For me it was very hard to read the titles and descriptions of the nodes because the titles would overlap, due to the complexicity of the graph. For that reason it took some time to find the correct nodes. (Maybe this was because I was using an IPad)."
- "Overlapping labels, very little use of colour, node clustering"
- "The orange text was not really visible
  No distinction between positive and negative besides text
  Hard to find Nodes"
- "It is visually cluttered.
  No scroll to zoom."
- "There is text everywhere, making it very hard to read or to find nodes. It was also hard to understand just by looking at the graph, what text belongs to which node. Adding relations also took rather long, since you have to select both nodes out of a long alphabetically ordered list. It is also incredibly difficult to understand correlations."
- "no color coded elements, lots of overlapping text, no search function"
- "Generally very confusing view. The text sometimes overlapped, making it unreadable. Without zooming in, it would be hard to understand."
- "zoom function is a bit confusing at the beginning
  sometimes difficult to find a specific node"
- "The Label of the nodes where to big to be displayed in the graph. Everything ovelaps, nothing can be seen."
- "yellow text sometimes hard to read, to much text on one note"
- "The visualization was very confusing and unclear. I was not able to get an overview. Distinguishing between different aspects like correlations and types of nodes was very hard. It was an information overload."

Question: What could be improved with the basic visualization?

- "Visual elements or features to improve the overview and navigation of the system "

- “Maybe force the graph drawing to spread out the labels more and select other colors/opacities for edges and labels, or overall improve the spacing on the entire web page (like using the full screen, flexbox + flex-grow + flex-col ;))
  also a search would be nice.
  Effect edges could be separated from relation edges, e.g. by colors, or the weight with edge thickness”
- “Put nodes more appart, labels should not overlap”
- “-Choose a smarer way to sort the nodes.
  - Mark different quality aspects (expl. colors)”
- “Maybe add a search feature, where the node that you are searching for, is for example highlighted.”
- “Add colour, increase node spacing”
- “More Colours for negative and positive correlation
  Labeling without <> and with spaces
  Find Nodes easily via search, not only through F5”
- “Less text overlap for better readability.
  Implement zoom by scrolling.”
- “Moving text, so that it doesn’t overlap and removing some of the text in favour of symbols. Improving the usability of adding and removing relations and nodes.”
- “a search function, colors, no overlapping text”
- “Less text in the overall view. Maybe comprehend details in every node itself, so by clicking on it u get more information.”
- “add search function for nodes”
- “make the graph visualization like in the advanced visualization :D”
- “give the tree branches more room so the nodes are not that cramped.”
- “Making it easier to understand by adding visual help to distinguish different types of nodes and relations.”

Question: What did you like about the advanced visualization?

- “I really liked the improvement of visualization and understanding by the graphical elements (e.g. colors and elements). Also the legend helped me understand and navigate the graph fast”
- “Colors! And overall visibility in the graph way better than in the basic mode.”

- "Manipulation of the graph without input masks"
- "The visualization gives a much better idea of which characteristics of the abstract quality aspects are related. Its good that you can see what aspects are corralated. By using colors you can distinguish very well different aspect."
- "I liked that the graph was using colored nodes and realtions to make the graph easier to read and understand.
  I liked the tree overview, because it gave you a clear list of all the elements and Relations.
  Filter was also very practical."
- "Coloured edges to indicate correlations, well–spaced nodes with useful colouring and clearly legible labels, a legend explaining their meaning, tree–table pop-up clearly showing the negative influence of the "effort of package construction""
- "Less Text for a better overview"
- "Colours for easy identification."
- "The color coding of nodes and relationships helped with understanding the graph. The text was easy to read. The Treetable especially was very helpful in understanding the relationships and being able to adjust values easily was also a good feature to understand the correlations. Adding relations just by klicking on the nodes is very intuitive."
- "the overview, the filter and the color coding, the sliders and adapting the values"
- "The legend on the side was very useful. In general, better overview, easier to understand and very user-friendly. The different colors helped with the clarity of the relations between nodes."
- "easy to understand
  Intuitive operation"
- "colors, readability, tree view"
- "more colors to see the different branches of the tree better, "
- "This visualization gave a clear overview without being distracted. Especially correlations were very easy to see and quick to understand."

Question: What did you not like about the advanced visualization?

- "To add a relation you need to select given nodes. In basic visualization you could just start off with the interface."
- "Still though quite synthetic visuals, and Bootstrap :(("

- “Sometimes hard to click on nodes”
- “-It is hard to use it without some expertise in the topic
  - the graph visualization is still a bit messy
  - interface does not always work
  - treetable view ☼
  - top bar is not fixed to the top and made with bootstrap”
- “The „add Relation“ Button did not open a window where i could select the nodes from a list. Instead I had to look for the nodes in the graph and click on them. I found the first option more easy to use.”
- “Navigation (especially zooming) is cumbersome”
- “Too colour reliant”
- “No scroll to zoom.”
- “The node that I added was hidden behind the Legend, so it took me a little to find it. As someone who never used a software like this before, it was not immediately clear in which order I had to klick on the nodes to make one a subclass of the other.”
- “searching has to be done manually”
- “That you would have to click on the nodes to connect them, and that there were no other options. This will get very hard in bigger projects to even find the nodes.”
- “zoom function is a bit confusing at the beginning
  sometimes difficult to find a specific node”
- “Adding relations is only possible by clicking on the graphical represenation of the node, not by entering their names”
- “.”
- “Metric and measure seem to be similar to positive and negative correlation because only the color differed.”

Question: What could be improved with the advanced visualization?

- “The ability to search for elements”
- “More spaceous graph drawing would be benefitial, and user interaction with the graph still lacks some intuition”
- “Node selection”

- "- bug fixing
  - simpler GUI
  - usability"
- "Maybe do the add relation button the same as it is in the basic visualization."
- "Gesture–based navigation (like a maps application) with scrolling holding down a key or pinching on the trackpad"
- "Find Nodes easily via search, not only through F5
  List all relations of a node in a table
  correlations not with colour, but with patterns like broken lines / bigger lines etc."
- "Implement zoom by scrolling."
- "In the overview, there are some cases where it is hard to differentiate if a relation is just a corrlation or a correlation and a subclass at the same time. Maybe arrows could be used to signify subclasses.

  With many nodes on screen, the difference between the highlighted node and the other nodes can be hard to see. A possible improvement for example could be having the pop up, that appears when hovering over a node, be displayed permanently, while a node is highlighted. It would also be nice if a node that was just added is automatically highlighted to make it easier to find it."
- "add a search function"
- "Maybe also add the option, from the basic visualization, to connect nodes by their names."
- "search function for nodes"
- "adding relations should be possible in the graphical way AND in the way it is possible in the basic visualization"
- "give the nodes a bit more room because some branches from different previous branches are overlapping. Give the opportunity to select one specific branch, eg. Usability and only see all branches going to usability and going from usability."
- "Adding a version for colorblind people like different colors or shapes for nodes and relations."

Additional feedback:

- "Bootstrap UI :(but otherwise nice job"
- "It is forgivable that certain aspects are not perfect, because at the time of creation no such software quality visualization existed and the development was more difficult :)"

- “Also, when I wanted A to be the subclass of B, I didn’t know whether I would have to put A first of B first. But apart from that I liked it!”
- “The difference in usability of the basic version compared to the advanced version is very stark. ”
- “A very good system that helps in getting an overview on others/your projects and maybe even get a better understanding.”
- “All in all the Advances version was better by a margin as getting an overview of the project is important, which the advanced version managed to do in contrast to the basic version. While working with the basic version seemed like a burden, working with the advanced visualization seemed comfortable.”

Bibliography

[11] *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Standard. Geneva, CH: International Organization for Standardization, 2011 (cit. on pp. 1, 7, 8, 16).

[18] *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. Standard. Geneva, CH: International Organization for Standardization, 2018 (cit. on p. 42).

[90] "IEEE Standard Glossary of Software Engineering Terminology." In: *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064 (cit. on p. 1).

[AH04] G. Antoniou and F. van Harmelen. "Web Ontology Language: OWL." In: *Handbook on Ontologies*. Ed. by S. Steffen and R. Studer. International Handbooks on Information Systems. Springer-Verlag, 2004, pp. 67–92. ISBN: 978-3-540-24750-0. DOI: https://doi.org/10.1007/978-3-540-24750-0 (cit. on p. 5).

[Bar+95] M. Barbacci et al. *Quality Attributes*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1995 (cit. on p. 18).

[BAT97] P. Borst, H. Akkermans, and J. Top. "Engineering ontologies." In: *International Journal of Human-Computer Studies* 46.2 (1997), pp. 365–406. ISSN: 1071-5819. DOI: https://doi.org/10.1006/ijhc.1996.0096. URL: https://www.sciencedirect.com/science/article/pii/S1071581996900968 (cit. on p. 3).

[Bec14] D. Becket. *RDF 1.1 N-Triples*. 2014. URL: https://www.w3.org/TR/2014/REC-n-triples-20140225/ (visited on 08/24/2022) (cit. on pp. 4, 5, 33).

[Bez+10] A. Bezerianos et al. "Graphdice: A system for exploring multivariate social networks." In: *Computer graphics forum*. Vol. 29. 3. Wiley Online Library. 2010, pp. 863–872 (cit. on p. 16).

[BG14] D. Brickley and R. Guha. *RDF Schema 1.1*. 2014. URL: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/ (visited on 09/02/2022) (cit. on p. 4).

[BKM08] A. Bangor, P. T. Kortum, and J. T. Miller. "An empirical evaluation of the system usability scale." In: *Intl. Journal of Human–Computer Interaction* 24.6 (2008), pp. 574–594 (cit. on pp. 42, 44).

[BKM09] A. Bangor, P. Kortum, and J. Miller. "Determining what individual SUS scores mean: Adding an adjective rating scale." In: *Journal of usability studies* 4.3 (2009), pp. 114–123 (cit. on p. 44).

[BNO13] S. Bellomo, R. L. Nord, and I. Ozkaya. "A study of enabling factors for rapid fielding combined practices to balance speed and stability." In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. 2013, pp. 982–991 (cit. on p. 31).

[Boe+07] B. Boehm et al. "Fifth workshop on software quality." In: *Companion of the 29th International Conference on Software Engineering*. 2007 (cit. on p. 7).

[Bøe08] J. Bøegh. "A new standard for quality requirements." In: *IEEE software* 25.2 (2008), pp. 57–63 (cit. on p. 1).

[Bro95] J. Brooke. "SUS: A quick and dirty usability scale." In: *Usability Eval. Ind.* 189 (Nov. 1995) (cit. on p. 42).

[Bur+22] B. Burns et al. *Kubernetes: up and running*. " O'Reilly Media, Inc.", 2022 (cit. on p. 33).

[Cha91] D. de Champeaux. "Object-oriented analysis and top-down software development." In: *ECOOP'91 European Conference on Object-Oriented Programming*. Ed. by P. America. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 360–376. ISBN: 978-3-540-47537-8 (cit. on p. 8).

[DB-22] DB-Engines. *DB-Engines Ranking of Graph DBMS*. 2022. URL: `https://db-engines.com/en/ranking/graph+dbms` (visited on 08/18/2022) (cit. on p. 32).

[Del+03] I. Deligiannis et al. "An empirical investigation of an object-oriented design heuristic for maintainability." In: *Journal of Systems and Software* 65 (Feb. 2003), pp. 127–139. DOI: `10.1016/S0164-1212(02)00054-7` (cit. on p. 30).

[El +20] A. El Bolock et al. "Visualizing Complex Ontologies Through Sub-Ontology Extraction." In: *2020 24th International Conference Information Visualisation (IV)*. IEEE, 2020, pp. 509–514. ISBN: 1728191343 (cit. on p. 11).

[Gam+95] E. Gamma et al. *Design patterns elements of reusable object oriented software*. Addison-Wesley professional computing series. Addison-Wesley, 1995. ISBN: 0201633612 (cit. on pp. 29, 30).

[GB95] V. S. Gordon and J. M. Bieman. "Rapid prototyping: lessons learned." In: *IEEE software* 12.1 (1995), pp. 85–95 (cit. on p. 8).

[GOS09] N. Guarino, D. Oberle, and S. Staab. "What Is an Ontology?" In: *Handbook on Ontologies*. Ed. by S. Staab and R. Studer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–17. ISBN: 978-3-540-92673-3. DOI: `10.1007/978-3-540-92673-3_0`. URL: `https://doi.org/10.1007/978-3-540-92673-3_0` (cit. on p. 3).

[Gra94] A. Grazebrook. "Rapid prototyping of software and hybrid systems." In: *Concurrent Engineering: Concepts, implementation and practice*. Ed. by C. S. Syan and U. Menon. Dordrecht: Springer Netherlands, 1994, pp. 151–159. ISBN: 978-94-011-1298-7. DOI: 10.1007/978-94-011-1298-7_9. URL: https://doi.org/10.1007/978-94-011-1298-7_9 (cit. on p. 8).

[Gru93] T. R. Gruber. "A translation approach to portable ontology specifications." In: *Knowledge Acquisition* 5.2 (1993), pp. 199–220. ISSN: 1042-8143. DOI: https://doi.org/10.1006/knac.1993.1008. URL: https://www.sciencedirect.com/science/article/pii/S1042814383710083 (cit. on p. 3).

[GS14] F. Gandon and G. Schreiber. *RDF 1.1 XML Syntax*. 2014. URL: https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/ (visited on 08/18/2020) (cit. on pp. 4, 18).

[Han10] E. Hanser. *Agile Prozesse: Von XP über Scrum bis MAP*. 1. Springer-Verlag, 2010. ISBN: 978-3-642-12313-9. DOI: https://doi.org/10.1007/978-3-642-12313-9 (cit. on p. 31).

[HP16] T. Hovorushchenko and O. Pomorova. "Evaluation of mutual influences of software quality characteristics based ISO 25010:2011." In: *2016 XIth International Scientific and Technical Conference Computer Sciences and Information Technologies (CSIT)*. 2016, pp. 80–83. DOI: 10.1109/STC-CSIT.2016.7589874 (cit. on pp. 1, 7, 8, 16).

[iso] iso25000.com. *ISO/IEC 25010*. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (visited on 08/22/2022) (cit. on pp. 40, 41).

[Jaz07] M. Jazayeri. "Some trends in web application development." In: *Future of Software Engineering (FOSE'07)*. IEEE. 2007, pp. 199–213 (cit. on p. 31).

[JGr22] JGraph Ltd. *jgraph/drawio*. 2022. URL: https://github.com/jgraph/drawio (visited on 09/23/2022) (cit. on p. 13).

[Jør04] M. Jørgensen. "A review of studies on expert estimation of software development effort." In: *Journal of Systems and Software* 70.1-2 (2004), pp. 37–60 (cit. on p. 8).

[KLR16] M. Kara, O. Lamouchi, and A. Ramdane-Cherif. "Ontology software quality model for fuzzy logic evaluation approach." In: *Procedia Computer Science* 83 (2016), pp. 637–641 (cit. on p. 1).

[KPW14] A. Kerren, H. C. Purchase, and M. O. Ward. "Introduction to multivariate network visualization." In: *Multivariate Network Visualization*. Springer, 2014, pp. 1–9 (cit. on p. 16).

[Kra21] H. Krasner. *The Cost of Poor Software Quality in the US: A 2020 Report*. Report. Consortium for Information & Software Quality™ (CISQ™), 2021 (cit. on p. 1).

[KRT06] C. R. Kothari, D. J. Russomanno, and P. N. Tran. "Interactive Elicitation of Relation Semantics for the Semantic Web." In: *Advances in Systems, Computing Sciences and Software Engineering*. Ed. by T. Sobh and K. Elleithy. Dordrecht: Springer Netherlands, 2006, pp. 47–52. ISBN: 978-1-4020-5263-7 (cit. on p. 12).

[Lee+06] B. Lee et al. "Task taxonomy for graph visualization." In: *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*. 2006, pp. 1–5 (cit. on p. 17).

[LL13] J. Ludewig and H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt.verlag, 2013 (cit. on pp. 1, 8).

[LR01] A. Leff and J. T. Rayfield. "Web-application development using the model/view/controller design pattern." In: *Proceedings fifth ieee international enterprise distributed object computing conference*. IEEE. 2001, pp. 118–127 (cit. on pp. 30, 31).

[McB04] B. McBride. "The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS." In: *Handbook on Ontologies*. Ed. by S. Steffen and R. Studer. International Handbooks on Information Systems. Springer-Verlag, 2004, pp. 51–66. ISBN: 978-3-540-24750-0. DOI: https://doi.org/10.1007/978-3-540-24750-0 (cit. on pp. 4, 5).

[MD06] E. Mnkandla and B. Dwolatzky. "Defining agile software quality assurance." In: *2006 International Conference on Software Engineering Advances (ICSEA'06)*. IEEE. 2006, pp. 36–36 (cit. on pp. 1, 7).

[MH04] D. L. McGuinness and F. van Harmelen. *OWL Web Ontology Language Overview*. 2004. URL: https://www.w3.org/TR/2004/REC-owl-features-20040210/ (visited on 08/24/2022) (cit. on p. 5).

[MK18] L. Madeyski and B. Kitchenham. "Effect sizes and their variance for AB/BA crossover design studies." In: *Empirical Software Engineering* 23.4 (2018) (cit. on p. 39).

[Mus15] M. A. Musen. "The Protégé Project: A Look Back and a Look Forward." In: *AI Matters* 1.4 (June 2015), pp. 4–12. DOI: 10.1145/2757001.2757003. URL: https://doi.org/10.1145/2757001.2757003 (cit. on p. 13).

[neoa] neo4j. *Cypher Query Language*. URL: https://neo4j.com/developer/cypher/ (visited on 09/23/2022) (cit. on p. 13).

[neob] neo4j. *Neo4j Browser*. URL: https://neo4j.com/docs/operations-manual/current/installation/neo4j-browser/ (visited on 09/23/2022) (cit. on p. 13).

[neoc] neo4j Labs. *neosemantics (n10s): Neo4j RDF & Semantics toolkit*. URL: https://neo4j.com/labs/neosemantics/ (visited on 09/14/2022) (cit. on p. 32).

[NSL18] C. Nobre, M. Streit, and A. Lex. "Juniper: A tree+ table approach to multivariate graph visualization." In: *IEEE transactions on visualization and computer graphics* 25.1 (2018), pp. 544–554. ISSN: 1077-2626 (cit. on pp. 1, 17, 23).

[ope] opencontainers. *OCI Image Format Specification*. URL: `https://github.com/opencontainers/image-spec` (visited on 08/22/2022) (cit. on p. 33).

[Par+12] C. Partl et al. "enRoute: Dynamic path extraction from biological pathway maps for in-depth experimental data analysis." In: *2012 IEEE Symposium on Biological Data Visualization (BioVis)*. IEEE. 2012, pp. 107–114 (cit. on pp. 16, 17).

[RBA17] B. B. Rad, H. J. Bhatti, and M. Ahmadi. "An introduction to docker and analysis of its performance." In: *International Journal of Computer Science and Network Security (IJCSNS)* 17.3 (2017), p. 228 (cit. on p. 33).

[Reu+90] L. H. Reuter et al. "Human perception and visualization." In: *Proceedings of the 1st conference on Visualization'90*. 1990, pp. 401–406 (cit. on p. 1).

[RRJ11] V. Ranwez, S. Ranwez, and S. Janaqi. "Subontology extraction using hyponym and hypernym closure on is-a directed acyclic graphs." In: *IEEE Transactions on Knowledge and Data Engineering* 24.12 (2011), pp. 2288–2300. ISSN: 1041-4347 (cit. on p. 12).

[SBF98] R. Studer, V. R. Benjamins, and D. Fensel. "Knowledge engineering: principles and methods." In: *Data & knowledge engineering* 25.1-2 (1998), pp. 161–197 (cit. on pp. 1, 3, 7).

[Sch13] N. Schilling. "Surveys and interviews." In: *Research methods in linguistics* (2013), pp. 96–115 (cit. on p. 39).

[Sha+06] J. S. Shaik et al. "3D Visualization of Relation Clusters from OWL Ontologies." In: *SWWS*. Citeseer, 2006, pp. 17–23 (cit. on p. 12).

[SHQ08] R. Shannon, T. Holland, and A. Quigley. "Multivariate graph drawing using parallel coordinate visualisations." In: *University College Dublin, School of Computer Science and Informatics, Tech. Rep* 6 (2008), p. 2008 (cit. on p. 17).

[SR06] J. Seidenberg and A. Rector. "Web ontology segmentation: analysis, classification and use." In: *Proceedings of the 15th international conference on World Wide Web*. 2006, pp. 13–22 (cit. on p. 12).

[Sun+20] Z. Sun et al. "Domain ontology construction and evaluation for the entire process of software testing." In: *IEEE Access* 8 (2020), pp. 205374–205385 (cit. on pp. 1, 7).

[Tea21] Team SimpleKPI. *What are Metrics and Measures – Explanation and Examples*. 2021. URL: `https://www.simplekpi.com/Blog/metrics-and-meassures-a-definitive-guide` (visited on 08/02/2022) (cit. on p. 18).