**NEOntometrics – A Public Endpoint For Calculating Ontology Metrics**

**Editor(s):** Name Surname, University, Country  
**Solicited review(s):** Name Surname, University, Country  
**Open review(s):** Name Surname, University, Country

Achim Reiz and Kurt Sandkuhl  
*Chair of Information Systems, Rostock University, 18051 Rostock, Germany*

**Abstract.** Ontologies are the cornerstone of the semantic web and knowledge graphs. They are available from various sources, come in many shapes and sizes, and differ widely in their attributes like expressivity, degree of interconnection, or the number of individuals. As sharing knowledge and meaning across human and computational actors emphasizes the reuse of existing ontologies, how can we select the ontology that best fits the individual use case? How to compare two ontologies or assess their different versions? Automatically calculated ontology metrics offer a starting point for a quality assessment. In the past years, a multitude of metrics have been proposed. However, metric implementations and validations for real-world data are scarce. For most of these proposed metrics, no software for their calculation is available (anymore). This work aims at solving this implementation gap. We present NEOntometrics, an open-source, flexible metric endpoint that offers (1.) an explorative help page that assists in understanding and selecting ontology metrics, (2.) a public metric calculation service that allows assessing ontologies from online resources, including git-based repositories for calculating evolutional data, with an (3.) adaptable architecture to adopt new metrics quickly. We further take a quick look at an existing ontology repository that outlines the potential of the software and show how to customize NEOntometrics to individual requirements.

**Keywords:** Ontology Metrics, Ontology Quality, Knoweldge Graph Semantic Web, owl, rdf

1. **Introduction**

Ontologies facilitate the shared understanding of a domain between people and systems [1]. They allow to structure and contextualize data to detect implicit knowledge, access this knowledge using complex queries, and integrate and interlink data from various sources while facilitating a common understanding.

Over time, the semantic web community developed countless ontologies. To give a perspective, the vocabulary repository “Linked Open Vocabulary (LOV)” [2] contains 773 ontologies. The portal “ontologypatterns.org” collects small, reusable ontology patterns and provides 236 artifacts. Bioportal [3], a large repository for biomedical ontologies, contains 996 ontologies. Moreover, many more ontologies are available on different sources like GitHub or private company repositories.

While the number of developed ontologies is extensive, there are just a few means available to assess and control the quality of these artifacts. For the development team that likes to integrate an ontology into their system, there is now way to quickly compare the main attributes of two or more different available ontologies that serve the same purpose. For the knowledge engineer, the missing assessment capabilities hinder the tracking of change impacts throughout the ontology lifetime.

As we will show in the next section, the lack of means for quality assessments does not originate from a lack of proposed metrics. Over time, a variety of ontology quality frameworks have been developed. What is missing are practical implementations of these metrics. Without a means to put these metrics into use, further empirical research cannot proceed, and the potential of ontology metrics remains theoretical.

*Corresponding author. E-mail: achim.reiz@uni-rostock.de*
This work aims at closing this gap by presenting a flexible, extensible metric calculation endpoint for RDF and OWL-based ontologies. The software enables metrics to be calculated and retrieved using a graphical user interface (GUI) or an application programming interface (API). It further aids users in learning about different metrics, their calculations, and possible interpretations through an interactive explorer for ontology metrics. If several versions of an ontology are available in a git-based repository, the development of the metric values over time can be tracked.

From a user’s perspective, we believe this software enables a bottom-up use of ontology metrics to control ontology development processes better. It allows an ontology manager to quickly understand in which direction his ontology evolves, identify disruptive changes, and ensure that an artifact adheres to self-given quality standards.

From a researcher’s perspective, NEOntometrics brings forward research on ontology metrics by providing us with large amounts of metric data. For example, this data allows us to study how ontologies evolve over their lifetime and test how ontology metrics perform in an actual application (e.g., [4,5]).

The paper is structured as follows: Section two summarizes the current state-of-the-art regarding ontology metrics and calculation software. Section three presents NEOntometrics with its architecture and usage of the API and GUI. In section four, we illustrate the use of NEOntometrics by presenting a case study showing what evaluations are possible using the service through a preliminary analysis of a calculated dataset. The final sections discuss the connection of ontology quality and metrics and conclude the paper.

2. Related Work

This section covers the previously published work relevant to this paper. Significant for our research are metric calculation proposals and ontology metric frameworks, covered in the first part of this section, and possible calculation implementations, covered in the second part.

There are many different evaluation methods available. Please note that we only consider criteria-based frameworks that allow for an automatic evaluation based on the structural attributes of the ontology. Metrics that need human intervention or additional input parameters are not considered relevant. That excludes evaluation methods based on a gold standard (additional input parameter is the task fulfillment level that an ontology can provide in a given context), or corpus-based (additional input parameters are domain-related documents like a text corpus). An overview of these evaluation categories can be found in [6].

2.1. Related Quality Frameworks

Lozano-Tello and Gómez-Pérez published the Ontometric framework in 2004 [7]. It proposes evaluation attributes in five criteria: (Development) tool, (ontology) language, context, methodology, and cost. Arguably, some metrics have become obsolete due to standardization in the past years. In 2004, the web ontology language (OWL) was just released, and other representations like OIL, DAML+OIL, and SHOE were still actively used. Here, Ontometric is targeted to make the influences of the languages explicit and comparable. Today though, regarding the category languages, RDFS-based ontologies can be considered state-of-the-art and are mostly compatible with each other.

Further, the standardization decoupled the tools from the ontology. Thus, the tool capabilities do not influence the semantic artifact. Other proposed elements, however, can be supported by an automatic calculation, like the metric maximum depth in the category content. While the relevance today might be somewhat limited, Ontometric considerably influenced the newer frameworks.

Gangemi et al. proposed an ontology evaluation framework based on a semiotic-meta-ontology O² that provides a formal definition for ontologies and their usage. Further, the authors define an ontology evaluation design pattern (oQual). Based on their O² definition, measurements assessing structural, task, corpus, and usability-based attributes are proposed [8]. A technical report by the same authors [9] further suggested 32 metrics in seven categories assessing mostly graph-related structures like depth, width, modularity, the distribution of siblings, or tangledness.

In 2005, Burton-Jones et al. presented the semiotic metric suite [10]. It comprises four main categories (syntactic, semantic, pragmatic, and social quality) and ten quality metrics. While some of these metrics are based solely on the structure of the ontology itself, others need further additional external information. Nine of these measurements, in theory, can be calculated automatically. Practically, some of the required data for some measures will probably not be available. Examples are the access count of the ontology or the
links from other ontologies to the currently assessed one.

OQuaRE was first proposed in 2011 by Duque-Ramos et al. [11]. It was since used in several publications, always involving the core team of the proposing authors. OQuaRE offers 19 calculable metrics and associates these metrics with quality dimensions like readability or accountability. Further, the framework ties metric results to quality ratings, thus providing an interpretation of the measurements. This holistic approach to quality is a unique characteristic among the frameworks.

However, during implementation, we experienced several heterogeneities in the metric definitions of the framework, with metrics with the same name by the same authors being defined differently in the publications. While we made efforts to align the metrics, our research show that their proposed quality scores fail to capture the ontologies that are developed by the community [4]. Our study was the first that has been made by authors that are not part of the team that proposed the framework; however, our application shall allow more thorough analysis in the future.

Tartir et al. published 19 metrics in the OntoQA framework in 2005 and 2007 [12,13]. While the framework does not provide a grading system for the metrics like OQuaRE, it aids the interpretation by describing how modeling decisions influence the metric results. Further, the authors propose measurements applicable not to the ontology as a whole but to the elements in an ontology. OntoQA also defines class- and relation-specific measurements. The relationship importance, for example, is calculated for each relation.

In his Ph.D. thesis, Vrandečić [14] gathered the state-of-the-art in ontology evaluation and introduced means to connect the various evaluation and improvement efforts. He condensed the quality proposals into eight quality characteristics (accuracy, adaptability, clarity, completeness, computational efficiency, conciseness, consistency, and organizational fitness) and proposed validation methods for these characteristics. These methods are primarily based on manual evaluation approaches. While he selected a few autonomous numeric calculation values, the thesis focuses primarily on providing methodologies for the knowledge engineer to improve ontologies in general without a particular focus on metrics.

2.2. Related Metric Calculation Software

As presented in the previous section, various metric frameworks have been developed over the past years. Some of these frameworks are merely theoretical in their proposals; others came with prototypical implementations. Further, tools that do not correspond to one of the proposed frameworks have been developed. The following section shows historical and current software for ontology metric calculation.

OntoKBEval by Lu and Haarslev [15] analyzes the structure of ontologies by providing graph-related measures like the number of levels or the number of concepts per level. The tool offers means to grasp clusters in the ontology and developed its own visualization “Xmas”-tree.

Tartir et al. developed a standalone java application for the OntoQA framework [13]. The application implements measures of the OntoQA framework, including metrics for the individual classes.

OntoCat, proposed by Cross and Pal [16], is a plugin for the Protégé editor and provides size- and structure-related metrics. They allow the assessment of the ontology as a whole but also provide metrics concerned with specific subsets of the given ontology.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Date</th>
<th>Type</th>
<th>Available</th>
<th>Open Source</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OntoKBEval</td>
<td>2006</td>
<td>S</td>
<td>No</td>
<td>No</td>
<td>[15]</td>
</tr>
<tr>
<td>OntoQA</td>
<td>2005</td>
<td>S</td>
<td>Yes¹</td>
<td>(No)²</td>
<td>[13]</td>
</tr>
<tr>
<td>OntoCat</td>
<td>2006</td>
<td>P+</td>
<td>No</td>
<td>No</td>
<td>[16]</td>
</tr>
<tr>
<td>S-OnetoEval</td>
<td>2008</td>
<td>S</td>
<td>No</td>
<td></td>
<td>[17]</td>
</tr>
<tr>
<td>Protégé</td>
<td>2015</td>
<td>S</td>
<td>Yes</td>
<td></td>
<td>[18]</td>
</tr>
<tr>
<td>OQuaRE</td>
<td>2018</td>
<td>WT, API</td>
<td>Yes</td>
<td>No</td>
<td>[4]</td>
</tr>
<tr>
<td>OntoKeeper</td>
<td>2017</td>
<td>WT</td>
<td>No</td>
<td>No</td>
<td>[19]</td>
</tr>
<tr>
<td>OOPS</td>
<td>2012</td>
<td>WT, API</td>
<td>Yes²</td>
<td>No</td>
<td>[20]</td>
</tr>
<tr>
<td>OntoMetrics</td>
<td>2015</td>
<td>WT, API</td>
<td>Yes³</td>
<td>No</td>
<td>[21,22]</td>
</tr>
</tbody>
</table>

¹ https://github.com/Samir-Tartir/OntoQA, The binary .jar files are available under CC license. The source code itself is not public.
² http://oops.linkeddata.es/
³ https://ontometrics.informatik.uni-rostock.de/,
S-OntoEval by Dividino et al. [17] serves as a calculation tool for, among others, the framework of Gagne et al. Its primary focus is on structural evaluation. However, the tool also calculates usability based on annotations and task performance based on ontology querying.

The Protégé editor [18] offers basic metrics on its landing page that counts the usage of owl-specific language constructs like the number of object property domain declarations or the number of classes.

The developers of the OQuaRE framework introduced a web tool to calculate their proposed metrics. It integrates a statistical correlation analysis of the metrics and a web service. Unfortunately, the tool suffers from the same issues as the framework [4], and the implemented metrics are heterogeneous and do not adhere to a clear definition.

Amith et al. developed the Semiotic-based Evaluation Management System (SEMS), later renamed OntoKeeper [19], which implements the semiotic suite by Burton-Jones et al.

The “OntOlogy Pitfall Scanner” (OOPS) by Poveda-Villalón et al. [20] approaches automatic ontology evaluation differently: They do not calculate ontology metrics but detect common modeling pitfalls like the use of is relationship instead of rdfs:subClassOf or wrongful equivalent relationships.

OntoMetrics, first developed by Lantow [22], is a web service for calculating several ontology metrics. It covers most of the OntoQA and oQual ontology metrics and integrates the owl-based axiom counts that are also part of Protégé. It was later extended with a web service by Reiz et al. [21].

2.3. The Need for Another Calculation Tool

As the previous section has shown, many frameworks and tools have been developed over the past years. That raises the question of whether a new calculation tool is necessary. We argue that our application fills essential gaps:

Missing Practicability. Most of the developed tools are no longer available. Even if they are available, their usability is often low. Many of the tools were used for the authors’ evaluation efforts and do not come with a state-of-the-art user interface or support a variety of ontology sources. Further, most of the software is not maintained. This problem is amplified by the fact that most of the software is:

Closed Source. None of the evaluated tools is fully open source. Not only hinders this reproducibility. It also prevents the community from maintaining the software and building on this previous research. If there is a need for another kind of evaluation, one has to start from scratch. We, thus, argue that the closed source leads to:

Isolation. The implementation efforts have stayed mainly isolated from one another. Hardly any tool has reached a broad acceptance within the community, and the ontology evaluation efforts of researchers using different tools are often not comparable. While there is a consensus that ontology evaluation is meaningful, there is no common understanding of how to do it.

3. NEOntometrics

NEOntometrics is the successor of the Ontometrics tool [21,22] (thus NEOntometrics). The old Ontometrics is one of the few ontology evaluation tools still available, but it does not scale well, provides fewer functionalities, sometimes redundant calculations, and is complicated to adapt [23].

The new tool NEOntometrics consumes git-based repositories, iterates through all of the commits (a commit is a published change in a repository), and calculates the metrics of the available ontology files. The software targets to solve many of the previously named challenges. It comes with a state-of-the-art user experience, a GraphQL endpoint, calculates various metrics, is quickly extensible through an ontology for creating and describing metrics, and is open source4. We believe it has the potential to become a community-accepted tool for calculating ontology metrics.

The following section details the software itself: it presents the different components of the service, how they interact, and the underlying development decisions. We also present how our ontology-based metric calculations are extensible for future usage. Afterward, we give an overview of how to put the software to use.

3.1. The Architecture of the Metric Calculation

One design goal was to create a flexible application for integrating new metrics. A researcher shall be able to adapt the application to their individual needs and quickly implement new required metrics.

---

4 https://github.com/achiminator/NEOntometrics – Upon acceptance, we will create a permanent ZENODO-DOI as well.
To achieve this adaptability, we did not encode all of the metrics of the various frameworks directly in the software but decomposed them into their building blocks. For example, the metric `axiom/class ratio` is not calculated at the time of the ontology analysis. Instead, their building blocks `axioms` and `classes` are saved in the database. The compositional values are then calculated at the time of querying.

The information on the ontology metrics is stored in an owl-based metric ontology. On startup of the application, multiple SPARQL-queries extract the codified knowledge and set up the backend and frontend. Thus, changing and adapting the ontology is sufficient to adapt the measures. [24] further details the underlying metric ontology.

Figure 1 presents an example of the metric elements in the ontology. **Elemental Metrics** contain the atomic measures that are used to build the compositions. For `Axiom Class Ratio`, the **Elemental Metrics** are `Axioms` (the number of axioms) and `Classes` (the number of classes). The ontology further specifies mathematical relationships between the metrics. In the given example, `Axiom Class Ratio` is the `subClassOf` (divisor only `Classes`) and (numerator only `Axioms`). The **Elemental Metrics** are connected to metric instances named identically to the implementation names in the calculation service and the elements in the database. In the example of the `Axioms`, this element has a relationship `implementedByName` `value axioms`.

All elements have rich annotations, providing human-centered meaning to the metrics. Additionally, some elements have links to further online resources or scientific publications. The annotations are the foundation for the **Metric Explorer**, where users find guidance on the available metrics.

New metrics that build upon the available **Elemental Metrics** can be created by modeling them in the ontology. Upon start, the application will make these custom metrics automatically available in the frontend and backend. Table 2 shows the frameworks that are already implemented in NEOntometrics and part of the **Metric Explorer** and the **Calculation Unit** at the time of publication. The case study in section 4.2 details how to create new frameworks, e.g., for individual metric frameworks in an organization.

<table>
<thead>
<tr>
<th>Name in NEOntometrics</th>
<th>Proposed By</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cohesion Metrics</td>
<td>Yao et. al</td>
<td>[25]</td>
</tr>
<tr>
<td>Complexity Metrics</td>
<td>Yhang, Zan, Ye</td>
<td>[26]</td>
</tr>
<tr>
<td>Good Ontology</td>
<td>Fernandez et. al</td>
<td>[27]</td>
</tr>
<tr>
<td>OQuaRE</td>
<td>Duque-Ramot et. al</td>
<td>[4,11]</td>
</tr>
<tr>
<td>OQual</td>
<td>Gangemi et al.</td>
<td>[8,9]</td>
</tr>
<tr>
<td>OntoQA</td>
<td>Tartir, Apinar</td>
<td>[12,13]</td>
</tr>
<tr>
<td>Complexity Cohesion</td>
<td>Orme, Yao, Etzkorn</td>
<td>[28]</td>
</tr>
</tbody>
</table>

Figure 1. The metric ontology (image adapted from [24])
3.2. The Architecture Of Application

The application is based on a dockerized micro-service architecture and consists of five components: the calculation-unit OPI (Ontology Programming Interface), the API, the worker application, a database for storing the calculated metrics, and a Redis interface for queueing jobs. The API and worker share a common codebase. Figure 3 depicts the interaction of the involved services.

The **frontend** contains the GUI. It is written using the multi-platform UI language **flutter** with its underlying client language **dart**. Upon loading, the frontend first queries the API for available ontology metrics based on the metric ontology. This data fills the help section **Metric Explorer**, which allows users to inform themselves about the various available metrics and the options for the calculation page. Afterward, the user can retrieve the requested ontology metrics or put them into the queue if they do not yet exist.

---

Figure 2. The process of analyzing and retrieving ontologies with NEOntometrics (without application startup)

---

5 https://flutter.dev/, https://dart.dev/
The **API** is the *django*-based⁶ endpoint for accessing already calculated metric data or requesting the analysis of new repositories. During the startup of the software, the application queries the metric ontology. It builds the frontend data and dynamically creates calculation code to provide the measurements of the frameworks that build upon the *Elemental Metrics*. After startup, a client can exploit GraphQL to check whether the data he requests exists already in the database. If so, he is able to retrieve all the selected metrics for a given repository. If not, it is possible to put the calculation of a given repository in the queue and track its progress.

The **worker** is responsible for the calculation of the metrics itself. It checks whether jobs are available in the *scheduler* Redis database. If that is the case, it starts the analysis by first cloning or pulling the git-repository, then iterating through every new file and commit, analyzing the owl ontologies using the OPI metrics endpoint. Afterward, the calculation results are stored in the *database*. The scheduling mechanism is based on *django-rq*.⁷ Even though the worker shares a code base with the API, it runs as a separate application. The number of parallel calculations can be scaled by increasing the number of workers.

The calculation service **OPI** is responsible for calculating metrics out of ontology documents. While it is based on the calculation service published in [21], most underlying code has been replaced. The old application struggled with ontology files larger than 10 MB due to inefficient memory allocation, had no separation of the calculation of the *elemental metrics*, and the *composed metrics* of the metric frameworks, and lacked support for reasoning. The old application was designed as standalone software, while the new calculation engine is hidden from the user and only accessed by the API.

The backend utilizes two languages: The **API** is written in python, and the calculation service **OPI** builds on Java. While the two languages add complexity to the application design, it allows the integration and use of the OWL API for ontology analysis⁸. This library provides many convenient functions for handling OWL and RDF-Files like an entity searcher or the automatic calculation of some axiom-based metrics. The Django web framework, in opposition, comes with great extensibility to build the application.

The calculation and retrieval process as a whole is depicted in Figure 2. At first, the frontend requests whether an ontology is already known in the system. Afterward, it either returns the queue information to the end-user or starts another request for the ontology metrics. At the same time, the worker applications and OPI work on the queued tasks.

### 3.3. The Metric Explorer

The page **Metric Explorer** is a dynamic help page of available metrics in NEOntometrics. The two main categories are *Elemental Metrics* and *Quality Frameworks*. The former contains the underlying atomic measurements of the ontologies. The authors of the software create all information shown in this category. *Quality Frameworks*, on the opposite, present the ontology quality metrics developed by other researchers, like the OntoQA Metrics [13] by Tartir et al., shown in section 2.2. Here, all information originates from the authors of the given frameworks.

The **Metric Explorer** provides information on five categories (though not all are filled for all the metrics). **Metric Definition** contains the formal definition of the metrics and how they are calculated, while **Metric Description** supplements a more human-readable explanation and, at times, an example. The **Metric**

---


⁷https://python-rq.org/patterns/django/

⁸http://owlcs.github.io/owlapi/
**Interpretation** guides practical usage. **Calculation** explains their decompositions into the **Elemental Metrics** using the metric names that are returned by the API, and **seeAlso** links to further resources like the corresponding papers or additional reads.

### 3.4. A Frontend for Humans

The tab **Calculation Engine** (as shown in Figure 5) is the main entry point for the metrics calculation. The end-user first selects the required metrics. Hovering over the elements shows additional information. Checking the box on the right toggles the calculation of all metrics of a given category. The “Already Calculated” button shows the calculated repositories that are stored in the database. While these repositories can be a starting point for further exploration, the user can also place a URL in the text box that points to a new git repository or location of an ontology file.

A click on the arrow starts the metric request. If the metric is unknown in the system, the application asks to queue the calculation task. If it is already in the queue, a notification informs of the progress. Once the data is analyzed, a click on the arrow leads to the metric results presented as a paginated table, representing the metric values for the different ontology versions. A drop-down menu in the header allows for selecting the various ontology files, and the download button exports the metrics into a .csv.

![Figure 5. Selecting ontology metrics in NEOntometrics](image)

### 3.5. An Interface for Machines

One goal of NEOntometrics is to allow the integration of ontology metrics into semantic web applications, which requires exposing the service using a standardized interface. Relevant open standards are **REST** and **GraphQL** for the web and **SPARQL** for querying the semantic web. NEOntometrics builds on **GraphQL**.

**GraphQ** together with **REST** has become a new de facto standard for sharing information on the web, and there is broad support in various programming languages and frameworks. This support includes the django web framework used in this project, where the graphene plugin\(^9\) allows utilizing the internal Object Relational Mapping. It allowed us to build the interface with comparatively little implementation effort, as the translation of requests to database queries is carried out automatically. While these integrations are also available for REST, **GraphQL** allows traverse relationships and precisely select attributes for querying. This avoidance of over-fetching is highly relevant for this use case, as one ontology version has over 100 ontology metrics, and the user likely selects just a few.

The **GraphQL** endpoint further provides documentation on the various available requests and possible return values, thus enabling the guided development of new queries. The interface is accessible through a browser on a GraphQL interface or any other GraphQL client.

**SPARQL**, as a graph-based query language, has similar attributes to **GraphQL** regarding relationship traversal and attribute selection. Additionally, proving a **SPARQL** endpoint would further allow the integration of the metric calculations into existing knowledge bases. Unfortunately, there is (currently) no support in the form of plugins for integrating such an endpoint into the used django framework. This lack makes the creation of such an endpoint dissimilar costlier.

### 4. Bringing NEOntometrics Into Use

The following presents application scenarios for the NEOntometrics application. The first case study shows the potential of analyzing ontology evolution. The second part presents possible integration scenarios for NEOntometrics.

\(^9\) [https://graphene-python.org/](https://graphene-python.org/)
4.1. Analyzing Ontology Evolution with NEOntometrics

Analyzing ontology metrics over time can tell a lot about underlying design decisions. The size of the changes indicates if an ontology evolves gradually or has disruptive changes, thus measures stability. They also allow to assess how attributes like the logical complexity (e.g., measures through the number of axioms that incorporate meaning), the coverage (e.g., measured through the number of classes or individuals), or the shape of the graph (e.g., measured through depth or breadth) change over time. These trends can aid ontology managers in deciding whether an artifact’s development goes in the right direction.

This case study analyzes the Evidence and Conclusion Ontology (ECO). ECO captures the biological coherences like "gene product X has function Y as supported by evidence Z" [29]. Please note that the NEOntometrics authors have no affiliation with the authors of the ontology nor with the biomedical field of research. Further, the goal of the section is not to evaluate quality but to observe the development of the ontology over time, to give an impression of possible assessments. Previous work discussed the connection between metrics and development choices from an ontology developer perspective [30].

The ECO repository has 856 commits in 17 ontology files. For this analysis, we were interested in the main ontologies in this repository. Thus, we only assessed the ontologies in the root structure, resulting in three ontology files. eco.owl with 89 versions, eco-basic.owl with 44, and eco-base.owl with 45 versions.

We first examined the axiom count of the ontologies and then used Tartir et al.’s OntoQA framework [12,13] for further analysis. The corresponding source files in Jupyter Notebooks are available online[10].

The first analysis is concerned with the development of the ontology size. Figure 6 presents the three ontology files in their different versions and plots the development of axioms with time. While the solid line represents all axioms overall, the dashed line only accounts for such that incorporate a logical meaning in rdfs or owl syntax. The difference between the dashed and the solid lines are, thus, annotations or custom-defined properties.

A first insight of the chart in Figure 6 is the variances of the logical axioms and the axioms in general. While the size of the ontology overall fluctuates intensely, the number of parts of the ontology that incorporates logical meaning stays relatively stable. One significant spike occurred between 2018 and 2019, which we will scrutinize further. Analysis reveals that a more extensive restructuring of the ontology drives this increase in logical axioms.

At first, the classes in eco doubled from around 900 to first a little over 2000, then further increased to over 3000. The number of defined object properties jumps from three to above 50, and the relation on classes through object properties increases from 350 to almost 2500, then drops to around 1600. This change event also marked the introduction of eco-base and eco-basic.

Figure 7 and Figure 8 show the relationship richness and schema deepness defined in the OntoQA framework. The former is the number of non-inheritance relationships divided by the sum of non-inheritance relationships and inheritance relationships; the latter is the number of subclasses per class [13]. They directly reflect the changes named above:

After an initial increase due to the rise in object properties, the relationship richness of eco drops with the increase in classes and subClassOf statements. Also, object properties were introduced. Later, a decline in object properties, combined with the further increase in classes and subClassOf statements, partially reverses the growth.

---

[10] Data: https://github.com/achiminator/NEOnto-Evaluation. Upon acceptance, we will upload the data permanently to Zenodo.
There are many more aspects that one could analyze for the given repository. For instance, we have only analyzed one file thoroughly and examined the impact of one main change event. As the last diagram already indicates, many more fluctuations are worth looking at. The variations affect the relationships between non-hierarchical and hierarchical relationships and classes and graph-related structures like the width or depth, individuals, or data properties. NEOntometrics provides the necessary tool to examine these questions further.

4.2. Adapting NEOntometrics by Adapting the Metric Ontology

A recent empirical analysis of 69 ontology evolution processes (based on NEOntometrics) has shown that the developments are highly heterogeneous and that assumptions on stereotypical development processes do not apply: There is no common rule or joint history that ontologies share [5].

If the ontologies are highly diverse, so is the required evaluation. This diversity of ontologies and their metrics emphasizes the careful selection of the latter. One person might build a taxonomy with rich human-readable annotations, and another targets to infer knowledge by modeling complex relationships between classes. A successfully applied metric by the first person might not work for the second.

While the metric explorer supports the selection process, a person might develop their measure to intertwine two metrics in a way that has not been done before. Organizations may want to select and reorder the metrics or limit the display to only relevant ones.

The following subsection explains how to adapt the application by altering the metric ontology. While every ontology editor can be utilized for editing, this section builds on the open-source software Protégé [18].

4.2.1. Restructuring the Ontology Metrics

The two classes Metrics and Quality_Frameworks are at the core of the metric calculation. The former represents the measurable ontology attributes and their implementation in NEOntometrics, and should only be changed if there is a need to define additional measurable ontology attributes. The Metrics subclasses are essential for the startup of the application, and their alteration or deletion can lead to undesired behaviors. Thus, the individualization effort should take place in the subclasses of the Quality_Frameworks.
Reusing the existing metrics is possible by creating new, individual subclass structures for dedicated purposes. After restart, the software reads the new structure and injects it into the code. As an effect, the frontend displays the new categories, and the new subgroups can be quickly selected, reducing the complexity for the metric consumer. The example of Figure 9 illustrates custom ordered metric categories in Protégé.

![Figure 9: Example of custom quality frameworks](image)

4.2.2. Creating New Ontology Metrics

The currently implemented quality frameworks based on the literature (cf. Table 2) already provide various metrics covering many use cases. Depending on the individual challenges, however, reusing the already existing metrics might not be sufficient. In these cases, creating custom metric classes can provide a possible solution.

The ontology provides formalized properties for the extension of the ontology. The annotation properties MetricDefinition, MetricDescription, and MetricInterpretation fill the respective fields in the metric explorer (cf. Figure 4) to help the metric consumer to select suitable measures.

![Figure 10: The formalized mathematical relationships for connecting the subclasses of Quality_Frameworks to Elemental_Metrics](image)

The object properties facilitate the connection of the reused or self-created Quality_Frameworks to the Elemental_Metrics and are the backbone for setting up the calculation unit. The subclasses of calculatedBy contain relations to describe the mathematical calculation operations of the application (cf. Figure 10).

The relation directlyUsesMetric states that a metric from a quality framework directly accesses an Elemental_Metric, e.g., the OQual.Absolute_Depth metric is a subclass of directlyUsesMetric only Total_Depth. Commutative operations like sum or multiplication are combined using the AND operator, e.g., sum only (Subclasses_Of_Thing and Super_Classes). Division and subtraction have further subclass for linking the elements. The mathematical relationships can be nested to create more complex queries.

As an example, the class with the name Average_Paths_Per_Concept, having a relationship SubClassOf (divisor only Classes) and (numerator only (sum only (Subclasses_Of_Thing and Super_Classes))) is first connected to the names of the given implemented database fields, represented by the connected individuals. Afterward, it is injected into NE-Ontometrics as:

\[
\text{rootClasses + superClasses} \\
\text{classes}
\]

5. Discussion: Measuring Ontology Quality Using Ontology Metrics

The apparent goal of a knowledge engineer is to develop high-quality ontologies. [31] collected three quality definitions relevant for ontology evaluation: The “conformance to requirements”, the “fitness to use”, and the “the totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs”. These quality definitions already emphasize the varying viewpoints of quality: Even though two persons use the same ontology, their requirements and, thus, their views on quality most likely differ.

At the same time, our empirical research has shown that the evolutional processes of ontologies are highly heterogeneous: There is no apparent stereotypical development process that ontologies follow [5]. While the OQuaRE framework defined a ready-to-use quality rating system for ontology metrics based on a school-like grading score, an extensive analysis showed that it does not adequately capture the ontologies being developed. Applying these out-of-the-box grades would likely lead to misleading results [4].

As a result, we do not believe that measuring ontologies using a standardized or arbitrary set of ontology metrics can help users create or select better artifacts. Based on his requirements, a user must first identify
the aspects that are important to him. Afterward, one can select the metrics that measure the selected attributes (e.g., using the Metric Explorer).

A single measurement point is different to interpret. For example, drawing conclusions from one measurement point of Average Path per Measurement Point (cf. section 4.2.2) is unlikely to yield beneficial results. However, they have explanatory power if they are used for comparison against other ontologies or versions. Comparing the metrics of two or more ontologies gives a reasonably good understanding of the differences in their structures. Grasping the (from a personal point of view) essential attributes quickly can speed up and improve reusing decisions. Moreover, as shown in section 4.1, comparing the versions tells much about the evolutionary process.

6. Conclusion

Ontologies are in use in various applications, facilitating meaning between human and computational actors and enabling these actors to harness the full potential of structured knowledge. The rising number of developed ontologies emphasize the need for quality control.

The research community has identified this need for quite some time. Many frameworks for controlling the quality of ontologies have been proposed, assessing several attributes of an ontology. However, implementations of these frameworks have been scarce. The missing software hinders the research progress: While the definition of measurements is important, it is then crucial to put these metrics into use.

The application proposed in this paper aims at closing this gap. We presented NEOntometrics, an open-source software to calculate ontology metrics. The application integrates several metric frameworks and is easily extensible. It is possible to analyze the development of metrics over time by analyzing git-based ontology repositories. Further, the user can inform themselves of available calculations and possible implications using an interactive metric explorer. The ontology metrics can be calculated and retrieved either using a graphical user interface or a GraphQL-API. While the former is targeted at knowledge engineers, the latter shall allow developers of semantic-based applications to integrate metrics into their software.

The preliminary case study in the previous section briefly presents possible applications. There are many more aspects worth looking at, like comparing typical development processes in different fields (e.g., industrial vs. biomedical ontologies), the usefulness of the proposed frameworks, and the modeling preferences of different persons, to name a few.

Further research will be concerned with analyzing the metric data itself. However, we will continue to work on the application, to add more metric sources and metrics on the specific elements within an ontology, like class-specific and relation-specific measurements, and integrate visualization capabilities. In this regard, we are interested in the aspects that the community would like to see implemented.

In the long term, we hope that NEOntometrics impacts the use and research of ontology metrics and believe it can help us empirically understand ontology modeling better.

References

and validation: An integrated formal model for the quality diagnostic task, Trentino, Italy, 2005.


