Test-driven knowledge graph construction

Jindřich Mynarz , Kateřina Haniková , and Vojtěch Svátek

Department of Information and Knowledge Engineering, Prague University of Economics and Business, Prague, Czech Republic
mynarzjindrich@gmail.com, {hank05,svatek}@vse.cz

2023

Abstract

We propose how to construct knowledge graphs using a method based on user requirements formulated as competency questions. It suggests to formalize the competency questions as SPARQL queries wrapped as SHACL constraints allowing them to be used as automated tests. The method defines a process to guide knowledge graph construction based on feedback from tests. It aims to reduce the engineering effort required to construct a knowledge graph that meets the requirements, while assessing the quality of semantic artifacts produced in this effort. We intend the method to provide a solid engineering basis for knowledge graph construction using the lessons learnt from software development and based on open standards. We demonstrate using the method on construction of a knowledge graph about antigen covid tests.

Knowledge graphs use graph-based data models to capture knowledge commonly combined from large and diverse data sources (Hogan et al. 2021). Constructing knowledge graphs can be an intricate and open-ended task that is in many ways an art rather than a craft.[1] While there are resources covering *how* to build knowledge graphs, there is little on *what* to build: how to elicit, formulate, and validate requirements for a knowledge graph. Overall, there is a shortage of solid engineering practices guiding through this task.

We propose how to construct knowledge graphs using a standards-based method founded upon user requirements formulated as competency questions (CQs). It suggests to formalize the CQs as SPARQL queries (Harris and Seaborne 2013) and wrap them as SHACL constraints (Holger Knublauch and Kontokostas 2017) in order to allow them to be used as automated tests. The method defines a process guiding knowledge graph construction based on feedback from tests. It aims to reduce the effort required to construct a knowledge graph that meets its requirements and passes quality assessment of semantic artifacts, such as ontologies, that are made in the process.

[1] The same argument was made about designing ontologies in Soldatova, Panov, and Džeroski (2016).

Motivation

Following a method for knowledge graph construction offers several benefits. In general, a method helps decide what to do next. In particular, it is useful in the beginning in helping to overcome the blank canvas paralysis. Moreover, a method breaks down the effort required for knowledge graph construction into sub-tasks, which can help organize and coordinate a team working on them.

The direction provided by a method anchored in user requirements can reduce the undirected upfront effort and help avoid overengineering. For example, it discourages needless effort spent on achieving a lossless transformation to the target knowledge graph that preserves all source data even though it might not be needed. It can help avoid premature abstraction and premature optimization, such as for performance or readability. Thanks to the tests checking if the user requirements are satisfied, we get an early proof of value instead of speculating about it.

Related work

When outlining this method, we can learn from ontology engineering, as it has a head start of several decades on knowledge graphs. There is a long tradition of using competency questions as requirements for ontologies (Gruninger and Fox 1994). Much of this experience can be reused, since ontologies serve as essential building blocks of knowledge graphs imbuing them with explicit semantics. Knowledge graphs in turn can be considered as ontologies populated with data. Creating tests for ontologies out of CQs is also nothing new Zemmouchi-Ghomari and Ghomari (2013). More recently, it was adopted for knowledge graphs too (Pan et al. 2017).

Broader still, we can adopt the practices established in software engineering. The hereby presented method borrows from the test-driven development cycle (Beck 2003), which is commonly characterized by the following steps:

- Add a test
- Run all tests, expecting the new test to fail
- Write the simplest code that passes the new test
- All tests should pass
- Refactor as needed

A fundamental feature of this cycle is that it intertwines development with testing, so that tests exercise the development artifacts and provide feedback informing further work. Tests provide a controlled way how to evolve the artifacts under development in response to changing requirements, such as when their scope is extended or they are refined iteratively. The field of ontology engineering is already adopting agile development approaches using tests. Ontology-specific

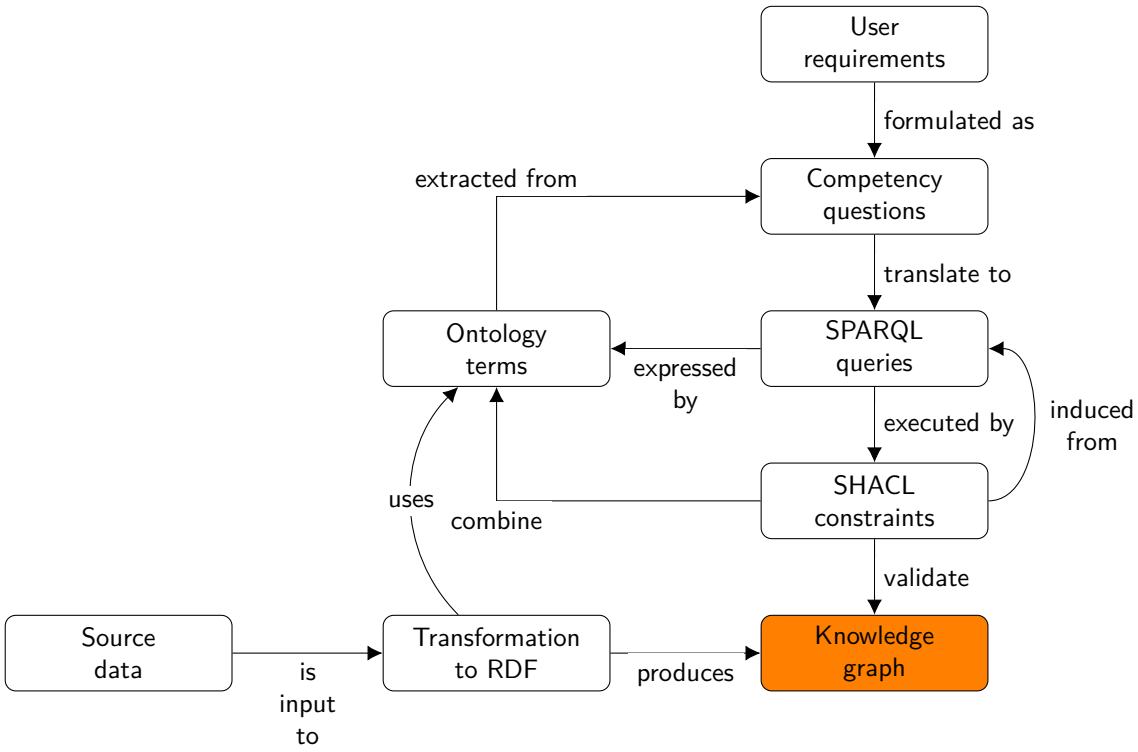

Figure 1: Knowledge graph construction method

methods, such as SAMOD (Peroni 2016) or Linked Open Terms (Poveda-Villalón et al. 2022), are being proposed.

The following text describes the proposed method for knowledge graph construction and demonstrates its use on a knowledge graph about antigen covid tests.

Method

We present a method for test-driven knowledge graph construction. The method proposes a sequence of steps and feedback loops between them. Each step produces or consumes one or more artifacts, such as ontologies or SPARQL queries. An overview of the relations between the key artifacts used by the method is depicted in Fig. 1. The method involves the following steps:

1. Start by using knowledge elicitation techniques (Shadbolt and Smart 2015) to gather user requirements. User requirements can be gathered from subject-matter experts or prospective users of the knowledge graph.

Formulate the user requirements as CQs (Gruninger and Fox 1994). CQs are analytical questions that the target knowledge graph is expected to answer. In order to avoid misinterpretation, they should be reviewed by those who expressed the requirements the CQs capture.

2. Analyze the CQs to extract terms and relations. CQs can be analyzed using the linguistic notion of presupposition, which can be defined as *"a condition that must be met for a linguistic expression to have a denotation"* (Ren et al. 2014). Viewed this way, CQs presuppose the terms with which they can be expressed. The terms to extract from CQs are typically nouns or noun phrases that refer to entities from the domain of the knowledge graph under construction. They can be considered lexical surface forms of these entities. The terms can be identified manually or with the help of tools, such as part-of-speech tagging. The relations to extract might be represented by verbs connecting the terms co-occurring in a CQ. Less direct mappings between entities and surface forms are possible too.

3. Formalize the terms and relations in a minimum viable ontology using RDF Schema (Brickley and Guha 2014). While terms typically map to classes (i.e. instances of `rdfs:Class`), relations map either to properties (i.e. instances of `rdf:Property`) or n-ary relations represented by classes. Document the ontology with definitions sourced from and validated by subject-matter experts to create a shared understanding. The ontology should make minimal ontological commitment, so avoid its upfront axiomatization, such as OWL restrictions. The resulting ontology shall serve as an explicit and machine-readable conceptualization of the knowledge graph's domain. Formalizing the ontology may in turn reveal ambiguity in the CQs. Ambiguous CQs cannot be formalized reliably since misinterpretations may occur. In such cases, sources of ambiguity should be revised with subject-matter experts aiming to reformulate the CQs in an unambiguous way.

4. Translate the CQs to SPARQL queries expressed by the ontology. SPARQL queries make the CQs executable. Here, we expect a manual translation of the CQs to SPARQL queries. Automated translation was attempted by others, such as in Espinoza-Arias, Garijo, and Corcho (2022). Start with syntactically valid SPARQL queries of hypothetical data expressed by the ontology. A CQ may translate to a SPARQL graph pattern composed of triple patterns joining the terms co-occurring in the question via specific relations. The way CQs should be translated to SPARQL queries largely depends on their expected answers. Ren et al. (2014) suggests that we might be *"more interested in checking if a CQ can be meaningfully answered, instead of directly answering a CQ."* When using such existential quantification, a SPARQL query formalizing a CQ can be expected to return non-empty results. What it means is that the query can answer a CQ using a given knowledge graph. It does not verify if the answer is correct. Conversely, some CQs may describe universal invariants of their

domain. Queries encoding these invariants are akin to property-based testing (Fink and Bishop 1997). In case stricter guarantees are needed, the expected correct answers can be included in the queries and validation constraints in SHACL implementing the CQs, corresponding to the usual example-based testing.

5. Wrap the queries as SPARQL-based SHACL constraints to automate their execution. SHACL defines a target[2] of each constraint. A target defines the scope of a CQ. Scope of the data graph validated by a given constraint may either encompass the entire knowledge graph or cover its subset. For instance, existentially quantified queries checking the complete data graph can target it by using `sh:targetNode []`. It is also possible to rewrite a part of a SPARQL query translating a CQ as target selection in SHACL. Alternatively, the prerequisite part of the query can be represented as a SPARQL-based target (Holger Knublauch, Allemang, and Steyskal 2017). An entity-centric partitioning can be implemented by defining a SPARQL-based target that covers a concise bounded description (Stickler 2005). Some data sources for the knowledge graph may natively partition data to independent subsets, such as API responses or messages from a queue, that can be effectively validated by specific subsets of SHACL constraints. SHACL also defines the criteria of correctness of the SPARQL queries. An `sh:ask` query is expected to return the boolean `true`, while any results produced by an `sh:select` query are treated as violations. Specific expected results can be either hard-coded into the queries or specified via `sh:hasValue` in case a single value is expected. When including the expected results in SPARQL-based constraints, we may run into the limitations of SHACL due to pre-binding of variables.[3] For example, the `VALUES` clause, which could represent results of `SELECT` queries, is not allowed.

6. Develop a transformation of the source data to the target knowledge graph that is expected to meet the constraints. This can be implemented in many ways, largely dependent on the format of the source data, so we will not cover it here. For an overview of these approaches, see e.g., Fensel et al. (2020).

7. Validate the knowledge graph under development with the SHACL constraints. If the validation fails, resolve the reported violations by fixing the artifacts created in the previous steps. Fix the most primary artifact causing the violations. For instance, data transformation shall not work around an insufficiently expressive ontology. Similarly, imposing a more expressive data model on the knowledge graph might reveal errors in its source data. In this way, the finer structure of its ontology makes the previously hidden data quality issues visible. Some feedback may even indicate ill-formed CQs in need of revision.

[2] https://www.w3.org/TR/shacl/#targets
[3] https://www.w3.org/TR/shacl/#pre-binding

8. Refactor the artifacts. The SHACL specification of the knowledge graph is a key artifact to refactor. It can serve as a formal specification of the anticipated use of the ontology. Moreover, SHACL can transcend a single ontology and specify how to combine terms reused from multiple ontologies. SHACL core constraints can be induced from SPARQL-based constraints to represent the frequently queried patterns in data. For instance, you can infer universally quantified axioms represented as SHACL constraints from a sample of existentially quantified axioms represented as SPARQL queries. In particular, SHACL can represent the expected relations in data. For example, more complex or distant relations might require the expressivity of SHACL to be represented, using property paths or SPARQL graph patterns. Even though SHACL does not support conditional constraints directly, they can be rewritten according to the inference rule of material implication $P \rightarrow Q \Leftrightarrow \neg P \vee Q$.

Non-functional requirements

Note that the above-described method tests if a knowledge graph meets the given user requirements. It does not evaluate how well these requirements are satisfied. Absence of errors does not imply that the knowledge graph is sound. Therefore, the functional user requirements should be complemented by non-functional requirements describing the desired qualities of the solution. These requirements can be checked and refactored by using alternative sources of feedback, such as:

Code review Elicit expert insight from ontology and data engineers.

User feedback The results produced by SPARQL queries formalizing the CQs can be judged to be incorrect by users. User feedback can identify incorrect interpretation of CQs or incorrect assumptions. It might prompt revisiting any of the previously described steps or lead to adding more CQs.

Usability Usability may manifest as developer experience of writing SPARQL queries for the knowledge graph. In particular, complexity and verbosity of SPARQL queries may be considered. In this way, usability of the queries indirectly reflects the usability of the ontology. For example, verbosity may be caused by duplicate data that can be abstracted to the ontology, while complexity can be reduced by introducing ontological shortcuts. Verbose SPARQL queries may also suggest a need for introducing abstraction or additional axioms into the ontology. In fact, some CQs can be answered directly using the knowledge formalized in the ontology, since SHACL expects the validated data graph[4] to include the ontologies it populates.

Performance Performance can be evaluated indirectly using the execution time of SHACL validation. This time is proportional to the execution times of the SPARQL queries the constraints include.

[4] https://www.w3.org/TR/shacl/#data-graph

Using the above-mentioned feedback is a balancing act of multi-objective optimization. For example, performance feedback may be incompatible with the feedback about verbosity. CQs may pose conflicting requirements. Addressing the feedback thus requires making informed trade-offs.

Limitations

As can be expected, the proposed method has several limitations. Here we recount some of them and suggest how to remedy them. The method may cause the developed artifacts outlined in Fig. 1 to overfit the user requirements in scope, which would hinder the reuse of the artifacts. This shortcoming can be remedied by including more CQs or focusing on reusability during refactoring. User requirements for the knowledge graph under construction can be more complex than what SPARQL can express. One way around it is formalizing the requirements in a more expressive programming language that extends SHACL, such as in Holgen Knublauch and Maria (2017). Ultimately, in order to ameliorate the limitations of this method, it is best combined with other methods, such as those for ontology design (e.g., Kendall and McGuinness (2019)).

Case study: Antigen covid tests knowledge graph

We used the described method to create a knowledge graph about antigen tests for SARS-CoV-2. Its source data was gathered to create an overview of different evaluations of selected rapid antigen tests available on the Czech market.[5] It was collected manually from several regulatory bodies, such as SÚKL[6] and EU HSC[7], and was combined with performance evaluations of the tests, such as their sensitivity and specificity, that came from independent studies from various countries. The data was collected into an XML file. We set to create a knowledge graph out of it to open it to a wider reuse and allow performing retrospective analyses of this data. All artifacts we developed in this effort, such as CQs, are available as open source.[8]

Understanding the source data and its domain was a key prerequisite of our work. The data covers 158 antigen SARS-CoV-2 tests and their evaluations from several sources. Each source had its way of describing the domain, which we needed to comprehend first to represent the source's data well and make it commensurable. We also needed to become aware of the purpose for which the data was collected and how it was represented to serve this purpose. The data

[5] https://covidtesty.vse.cz/english/test-evaluation-older-data/
[6] https://www.sukl.cz/prehled-testu-k-diagnostice-onemocneni-covid-19
[7] https://health.ec.europa.eu/health-security-and-infectious-diseases/crisis-management/covid-19-diagnostic-tests_en
[8] https://github.com/KIZI/antigen-covid-tests-knowledge-graph

was originally structured in XML for display on a web page, so we needed to map its visual encoding into semantics. Knowing the source data well, we could begin with the knowledge graph construction.

The first step of the proposed method is gathering user requirements. So, we started with capturing user requirements formulated as CQs, such as *"What is the sensitivity of given tests according to their manufacturers?"* We came up with 19 CQs in total, divides into two categories based on their objective. The first category of questions captured what information should be present in the knowledge graph, the second category included analytical questions.

The second step of the method is to analyse the CQs to extract terms and relations. Thus, we analysed the CQs and extracted terms and relations, such as *"sensitivity"*, *"test"*, or *"has manufacturer"* from the given example question; see example 1. Since we did not have many CQs, we extracted the terms and relations manually.

Listing 1 Example ontology terms

```
:DiagnosticSensitivity a rdfs:Class ;
  rdfs:label "Diagnostic sensitivity"@en .

:AntigenCovidTest a rdfs:Class ;
  rdfs:label "Antigen covid test"@en .

:hasManufacturer a rdf:Property ;
  rdfs:label "Has manufacturer"@en .
```

The third step was to create a minimum viable ontology from the extracted terms and relations, we did it mostly by reusing existing ontologies and using RDF Schema. For what we did not find suitable terms to reuse, we formalised terms in the simple Antigen Covid Test Ontology. This ontology aimed to allow the expression of the CQs in SPARQL. This could be an advantage of our proposed method, there is no need to create complex, complicated ontologies, just a few terms that cannot be reused from existing ontologies.

Having the minimum viable ontology, we moved to step four: translating the CQs into SPARQL queries. After this step, we approached step number five and wrapped the SPARQL queries as SHACL constraints, such as in the example 2.

As the sixth step, we started with implementing a transformation of the source data to RDF. We converted the input XML data into RDF/XML via an XSL transformation followed by SPARQL Update operations for post-processing. The resulting data was tested by the CQs implemented in SHACL. We automated the data processing and test execution by a script based on Jena command-line tools.[9] Given that this is a small knowledge graph of around 10 thousand RDF triples, we validated it as a whole.

[9] https://jena.apache.org/documentation/tools

Listing 2 Example competency question in SHACL

```
:CQ12 a sh:ConstraintComponent ;
  rdfs:label "Manufacturer-declared test sensitivity"@en ;
  sh:parameter [
    sh:path :cq12
  ] ;
  sh:nodeValidator [
    a sh:SPARQLAskValidator ;
    sh:message """What is the sensitivity of given tests
                  according to their manufacturers?"""@en ;
    sh:prefixes :prefixes ;
    sh:ask """
    ASK {
      $test a act:AntigenCovidTest ;
        schema:manufacturer ?manufacturer .

      [] a ncit:C41394 ;
        dcterms:subject $test ;
        dcterms:creator ?manufacturer ;
        rdf:value ?sensitivity .
    }
    """
  ] .
```

The final step of the proposed method is about refactoring artifacts. Our development work proceeded in several iterations guided by continuous feedback from the script for data transformation and validation. When the knowledge graph satisfied the CQs, we continued with refactoring the semantic artifacts we created. For example, we wanted to avoid blank nodes to make the process deterministic, so we improved the XSL transformation to generate IRIs instead. We also abstracted a parent class for evaluations in the ontology and adjusted the SHACL shapes accordingly.

Some errors were not detected by the tests based on CQs. For example, when querying the knowledge graph, we found that the data transformation to RDF mistakenly created IRIs of manufacturers based on their antigen covid test identifiers. Consequently, the resulting data related each manufacturer to one antigen covid test, so that it was not possible to group multiple tests by the same manufacturer. A common response to finding errors not covered by tests is improving tests coverage to enable detecting the errors found and thus avoid future regressions. We followed this practice and added a CQ comparing a given manufacturer's antigen covid tests. The CQ presupposed that there are manufacturers offering more than one test and would therefore fail in case of the above-described error. Apart from fixing the way we generated manufacturers' IRIs, we spent further effort on de-duplicating manufacturers via post-processing

by SPARQL Update operations.

While we had CQs about manufacturers' tests, the implementations of these CQs generally asked if manufacturers' tests matching certain criteria exist. Being encoded as SPARQL ASK queries, any non-empty query results satisfied these CQs. They did not detect when the results were not the expected ones, such as when not showing all manufacturer's tests in the example of duplicate manufacturers above. We addressed this limitation by asking a more specific CQ presupposing more specific assertions. The more specific assertions we make about our domain, the better we can detect invalid data describing it. Yet when the domain changes or when our understanding of the domain is flawed, such assertions are more likely to become invalid. Consequently, there is a trade-off to be made between an assertion's specificity and its durability in face of change.

We encountered several other challenges during the development of the knowledge graph, such handling the structure of the source data originally designed for a web presentation or frequent changes in relevant legislation and evaluation studies. Since the source data was collected manually, it was inconsistent and required fixes. Some of these inconsistencies manifested as duplicates and were detected by our test suite. Future work on this knowledge graph can be aimed at automatic updates of the data and expanding its coverage beyond the Czech market for antigen covid tests.

Conclusions

The hereby proposed method guides through the open-ended process of knowledge graph construction. It breaks the process down into a well-defined sequence of steps, feedback loops between them, and semantic artifacts that are produced or consumed in the process. The central contribution of the method is allowing to test if the produced knowledge graph satisfies the requirements for its construction. This is done via competency questions formulated as SPARQL queries embedded in SHACL shapes for test automation. As such, one of its key advantages is being based on established semantic web standards.

We share the experience of applying the method to build a knowledge graph about antigen covid tests. The method aided in collaborative development of this knowledge graph, allowing its incremental refinement without regressions. Since constructing knowledge graphs from the same user requirements using multiple methods, while comparing their outcomes, is prohibitively expensive, we expect future improvements of the proposed method to come from its applications on knowledge graphs with different requirements.

Acknowledgements

This research was supported by CHIST-ERA within the CIMPLE project (CHIST-ERA-19-XAI-003).

References

Beck, Kent. 2003. *Test-Driven Development: By Example*. Addison-Wesley.

Brickley, Dan, and Ramanathan V. Guha, eds. 2014. "RDF Schema 1.1." W3C Recommendation. https://www.w3.org/TR/rdf-schema.

Espinoza-Arias, Paola, Daniel Garijo, and Oscar Corcho. 2022. "Extending Ontology Engineering Practices to Facilitate Application Development." In *Knowledge Engineering and Knowledge Management*, edited by Oscar Corcho, Laura Hollink, Oliver Kutz, Nicolas Troquard, and Fajar J. Ekaputra, 19–35. Cham: Springer.

Fensel, Dieter, Umutcan Şimşek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. 2020. "How to Build a Knowledge Graph." In *Knowledge Graphs: Methodology, Tools and Selected Use Cases*, 11–68. Cham: Springer. https://doi.org/10.1007/978-3-030-37439-6_2.

Fink, George, and Matt Bishop. 1997. "Property-Based Testing: A New Approach to Testing for Assurance." *SIGSOFT Software Engineering Notes* 22 (4): 74--80. https://doi.org/10.1145/263244.263267.

Gruninger, Michael, and Mark S Fox. 1994. "The Design and Evaluation of Ontologies for Enterprise Engineering." In *Workshop on Implemented Ontologies, European Conference on Artificial Intelligence (ECAI)*.

Harris, Steve, and Andy Seaborne, eds. 2013. "SPARQL 1.1 Query Language." W3C Recommendation. http://www.w3.org/TR/sparql11-query.

Hogan, Aidan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, et al. 2021. "Knowledge Graphs." *ACM Computing Survey* 54 (4). https://doi.org/10.1145/3447772.

Kendall, Elisa F., and Deborah L. McGuinness. 2019. *Ontology Engineering*. Vol. 18. Synthesis Lecture on the Semantic Web: Theory and Technology. Morgan; Claypool.

Knublauch, Holgen, and Pano Maria, eds. 2017. "SHACL JavaScript Extensions." W3C Working Group Note. https://www.w3.org/TR/shacl-js.

Knublauch, Holger, Dean Allemang, and Simon Steyskal, eds. 2017. "SHACL Advanced Features." W3C Working Group Note. https://www.w3.org/TR/shacl-af.

Knublauch, Holger, and Dimitris Kontokostas, eds. 2017. "Shapes Constraint Language (SHACL)." W3C Recommendation. https://www.w3.org/TR/shacl.

Pan, Jeff Z., Nico Matentzoglu, Caroline Jay, Markel Vigo, and Yuting Zhao. 2017. "Understanding Author Intentions: Test Driven Knowledge Graph

Construction." In *Reasoning Web: Logical Foundation of Knowledge Graph Construction and Query Answering: 12th International Summer School 2016, Aberdeen, UK, September 5–9, 2016, Tutorial Lectures*, edited by Jeff Z. Pan, Diego Calvanese, Thomas Eiter, Ian Horrocks, Michael Kifer, Fangzhen Lin, and Yuting Zhao, 1–26. Cham: Springer. https://doi.org/10.1007/978-3-319-49493-7_1.

Peroni, Silvio. 2016. "A Simplified Agile Methodology for Ontology Development." In *OWL: Experiences and Directions – Reasoner Evaluation*, 55–69. Cham: Springer.

Poveda-Villalón, María, Alba Fernández-Izquierdo, Mariano Fernández-López, and Raúl García-Castro. 2022. "LOT: An Industrial Oriented Ontology Engineering Framework." *Engineering Applications of Artificial Intelligence* 111. https://doi.org/https://doi.org/10.1016/j.engappai.2022.104755.

Ren, Yuan, Artemis Parvizi, Chris Mellish, Jeff Z. Pan, Kees van Deemter, and Robert Stevens. 2014. "Towards Competency Question-Driven Ontology Authoring." In *The Semantic Web: Trends and Challenges*, edited by Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai, 752–67. Cham: Springer. https://link.springer.com/content/pdf/10.1007/978-3-319-07443-6_50.pdf.

Shadbolt, Nigel R., and Paul R. Smart. 2015. "Knowledge Elicitation." In *Evaluation of Human Work*, edited by John R. Wilson and Sarah Sharples, 4th ed., 163–200. Boca Raton: CRC Press.

Soldatova, Larisa, Panče Panov, and Sašo Džeroski. 2016. "Ontology Engineering: From an Art to a Craft." In *Ontology Engineering*, edited by Valentina Tamma, Mauro Dragoni, Rafael Gonçalves, and Agnieszka Ławrynowicz, 174–81. Cham: Springer.

Stickler, Patrick. 2005. "CBD: Concise Bounded Description." {W3C} Member Submission. W3C. https://www.w3.org/Submission/CBD.

Zemmouchi-Ghomari, Leila, and Abdessamed Réda Ghomari. 2013. "Translating Natural Language Competency Questions into SPARQL Queries: A Case Study." In *The First International Conference on Building and Exploring Web Based Environments*, 81–86.