The New DBpedia Release Cycle: Increasing Agility and Efficiency in Knowledge Extraction Workflows

Marvin Hofer\textsuperscript{1}, Sebastian Hellmann\textsuperscript{1} [https://global.dbpedia.org/id/3eGWH], Milan Dojchinovski\textsuperscript{1,2}[0000--0003--4844--4260], and Johannes Frey\textsuperscript{1}

\textsuperscript{1} Knowledge Integration and Language Technologies (KILT/AKSW) DBpedia Association/InfAI, Leipzig University, Germany lastname@informatik.uni-leipzig.de
\textsuperscript{2} Web Intelligence Research Group FIT, Czech Technical University in Prague milan.dojchinovski@fit.cvut.cz

Abstract. Since its inception in 2007, DBpedia has been constantly releasing open data in RDF, extracted from various Wikimedia projects using a complex software system called the DBpedia Information Extraction Framework (DIEF). For the past 12 years, the software has received a plethora of extensions by the community and produced releases with over 14.4 billion triples in 2016. Due to the increase of size and complexity the release process was facing many delays thus impacting the agility of the development. In this paper, we describe the new DBpedia release cycle including our innovated release workflow, which allows development teams (in particular those who publish large, open data) to implement agile, cost-efficient processes and scale up productivity. To address these challenges, the DBpedia release workflow has been redesigned so that its primary focus now is on productivity and agility while the quality aspect is assured by implementing a comprehensive testing methodology. We run experimental evaluation and we argue that the implemented measures increase agility and allow for cost-effective quality-control and debugging and thus achieve a higher level of maintainability. As a result of our innovation, DBpedia now produces on average over 13.5 billion triples per month with a minimal publishing effort.

Keywords: DBpedia · knowledge extraction · data publishing · quality assurance

1 Introduction

Since its inception in 2007, the DBpedia project [5] has been continuously releasing large, open datasets in RDF, extracted from Wikimedia projects such as Wikipedia and Wikidata. The data has been extracted using a complex software system known as the DBpedia Information Extraction Framework (DIEF) which over the past 12 years has received a plethora of extensions and fixes by the community. This resulted in creating large, monolithic releases with more than 14.4
billion triples (2016-10 release). Until 2017, The DBpedia release process has been primarily focused on data quality and size, however, it neglected two other important and desirable goals: productivity and agility (cf. [3] for balancing the magic triangle on quality, productivity and agility; this paper is a consequent extension of our previous work). Due to the sole focus on quality and the increased size and complexity, the release process was facing huge delays (from 12 to 17 month cycle) with an increasing costs of development and lower productivity. The releases were so large and complex that the DBpedia core team failed to produce them for almost 3 years (2017–2019). Note that it was not a performance nor scalability related issue. In order to address these challenges, the DBpedia release workflow has been redesigned so that its primary focus now is on productivity and agility while the quality aspect is assured by implementing a comprehensive testing methodology.

In this paper, we describe the new DBpedia release cycle including our innovated release workflow, which allows development teams (in particular those who publish large, open data) to implement agile, cost-effective processes and scale up productivity. As a result of our innovation DBpedia now produces on average over 13.5 billion triples per month with a minimal publishing effort.

The paper is organized as follows. First, in Section 2, we summarize the two biggest challenges as a motivation for our work, followed by an overview of the release workflow described in Section 3. The main process innovations and conceptual design principles are described in Section 4. The implemented testing methodology is described in Section 5 and the results from several experiments showing the impact, capabilities and the gain from the new release cycle are presented in Section 6. Section 7 reports on technologies that relate to ours. Finally, Section 8 concludes the paper and presents future work directions.

2 Background and Motivation

1. Agility. Data quality is one of the largest and oldest topics in computer science independent of current trends such as Big Data or Knowledge Graphs and has an incredible amount of facets to consider [8]. Data quality, often defined as “fitness for use”, poses many challenges that are frequently neglected or delayed in the software engineering process of applications until the very end, i.e. when the application is demonstrated to the end-user. In this paper, we will refer to this phase of the process as the “point-of-truth” since it marks an important transition of data (transferred between machines and software) to information. At this point, results are presented in a human-readable form so that humans can evaluate them according to their current knowledge and reasoning capacity. We argue that any delay or late manifestation of such a point-of-truth impacts cost-effectiveness of data quality management and stands in direct contradiction to the first and other principles of the agile manifesto “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.” [2]. Our release cycle, counteracts the delay by frequent, fixed time-based releases
in combination with automating the delivery of data to applications via the DBpedia Databus (cf. 4.1).

2. Efficiency. We focus on efficiency as a major factor of productivity. Data quality follows the Law of Diminishing Returns [7] (similar to Pareto-Efficiency or 80/20 rule), meaning that initially decent quality can be achieved easily, because many gaps or errors can be fixed efficiently. After a certain growth, data projects face diminishing returns where errors become increasingly much harder to find and fix, up to a point where adding more resources (e.g. human labor or development power) produces negative returns\(^3\). In our experience, there is no exception to the law of diminishing returns in data and it affects all data projects, be they manual such as Wikidata, semi-automatic such as DBpedia or fully automated machine learning approaches. Our second main lesson learned is that **data quality does not depend primarily on the effort invested** (e.g. by a large community) but on the efficiency of the development process and the capacity to effectively improve data in a sustainable manner. Measures to increase efficiency are traceability of errors (Section 4.2) combined with testing (Section 5).

3 DBpedia Release Cycle Overview

The DBpedia release cycle is a time-driven release process triggered on a regular basis (i.e. monthly). At a given point of time the DIFE framework (in a distributed computational environment) is executed and data is extracted on the latest Wikipedia dump. The core of the release cycle is the DBpedia Databus platform which acts as a data publishing middleware and is responsible for maintaining information about published data groups and artifacts.

**Data groups and artifacts.** The process creates five core data groups, each generated by a different extraction process\(^4\): i) generic—information extracted by the generic extractors, ii) mappings—information extracted using user specified...
Table 1. Size metrics (i.e. triples count) for DBpedia data groups and release periods.

| Version       | Generic    | Mappings   | Text        | Wikidata    |
|---------------|------------|------------|-------------|-------------|
| 2016.10.01    | 4,524,347,267 | 730,221,071 | 9,282,719,317 | 738,329,191 |
| 2019.08.30    | 4,109,424,594 | 953,187,791 | -           | -           |
| 2020.04.01    | 3,736,165,682 | 1,075,837,576 | 11,200,431,258 | 4,998,301,802 |

mapping rules, iii) text—extracted Wikipedia article’s content and iv) ontology—the DBpedia ontology and v) wikidata—extracted and mapped structured data from Wikidata. Each data group consists of one or more versioned data artifacts which represent a particular dataset in different formats, content (e.g. language) and compression variants. In other words, an artifact has versions that are a collection of multiple files.

**Publishing agents.** A publishing agent acts on behalf of a person or organization and contributes and publishes data on the Databus. The initial set of data groups are published on the Databus by the MARVIN publishing agent. In addition to the MARVIN agent, there is also the DBpedia agent, which publishes cleaned data artifacts, i.e. syntactically valid and the configuration files used to generate the MARVIN and DBpedia releases are available as public git repository.

**Cleansing, validation and reporting.** The data published by the MARVIN agent is then picked up, the data (i.e. triples) is parsed so that only valid RDF and error reports are created. Finally, syntactically cleaned data artifacts are published by the DBpedia agent. While the data might be syntactically valid, yet it does not mean that there are no issues with the release. For example, the IRIs of particular subjects, predicates and objects should conform to a predefined schema, the data should be structurally correct and conform to the ontology restrictions, or the release might be incomplete (e.g. missing artifacts). In order to check the release for such issues, after each release a large-scale validation is executed and the errors reported. These reports are then delivered to the community for a review. The community reports issues which are considered for the next release cycle. Figure 1 depicts the overall DBpedia data release cycle.

The new DBpedia release approach has been deployed recently and releases are created on monthly basis (text every third month). Table 1 provides statistics for different DBpedia data groups for three releases; from Oct 2016, Aug 2019 and Apr 2020. 2016-10 is the last monolithic legacy release, which we added for comparability. Note that we do not provide numbers for ‘text’ and ‘wikidata’ data groups for the ‘2019.08.30’ since the incompleteness of these releases.

The numbers from Table 1 show that the number of triples for the ‘mappings’, ‘text’ and ‘wikidata’ data groups is constantly increasing over time. By

---

5 https://databus.dbpedia.org/marvin/
6 https://git.informatik.uni-leipzig.de/dbpedia-assoc/marvin-config
7 https://databus.dbpedia.org/vehnem/collections/dbpedia-2016-10-01
8 https://databus.dbpedia.org/vehnem/collections/dbpedia-2019-08-30
9 https://databus.dbpedia.org/vehnem/collections/dbpedia-2020-04-01
contrast, the 'generic' data group provides less triples. This is primarily due to the strict testing procedures which has been enforced, which caused, on one hand, lower number of statements but data of higher quality on the other hand. Also, the applied 'generic' and 'wikidata' extraction configuration varies in the number of enabled extractors. Compared to the Wikidata statistic\textsuperscript{10}, the DBpedia 'wikidata' extraction produces five times the amount of statements published by itself, mainly because of reification and materialization processes during the extraction (e.g. transitive instance types).

4 Conceptual Design Principles

The design and implementation of the new DBpedia release cycle has been driven by two design principles: i) time-driven data releases-enable more-frequent and regular time-driven DBpedia releases, and ii) traceability and issue management-enable more efficient management of issues and tracing of errors.

4.1 Time-driven vs. quality-driven data releases

While many of the principles of the agile manifesto are applicable here, the most relevant principle "Working software is the primary measure of progress"\textsuperscript{2} can not be applied directly to data. As motivated in Section 2, the judgment whether "data is working" is withheld until the "point-of-truth" at customer/end-user side. From our own past experience and from conversations with related development teams, it is a fallacy that the developer or data publisher has the capacity to evaluate when "data is useful", following their own quality-driven or feature-driven agenda. Since adopting an attitude of "quality creep"\textsuperscript{11} bears the risk of delaying releases and prevent data reaching end-users with valuable feedback, we decided to switch to a strict time-based schedule for releasing following these principles:

1. **Automated schedule vs. self-discipline.** Releases are fully automated via the MARVIN extraction robot. This alleviates developers from the decision when "data is ready". Else extensive testing of data might have the adverse effect. Developers are prone to "fixing one more bug" instead of delivery and proper end-user feedback.

2. **Subordination of software.** the whole software development cycle is completely subordinate to the data release cycle with time-driven, automatic check-out of the tested master branch.

3. **Automated delivery.** data is published on the DBpedia Databus, which allows subscription of data (artifacts/versions/files) via SPARQL and collections, which in turn enables auto-updated application deployment\textsuperscript{12} and therefore facilitating point-of-truth feedback opportunities earlier and continuously.

\textsuperscript{10} https://tools.wmflabs.org/wikidata-todo/stats.php

\textsuperscript{11} analogous to feature creep in software

\textsuperscript{12} via Docker, out of scope for this paper, see https://wiki.dbpedia.org/develop/datasets/latest-core-dataset-releases
4.2 Traceability and Issue Management

Any data issues discovered at the point-of-truth, starts a costly process of backtracking the error in reverse order of the pipeline that delivered it. The problem of tracing and fixing errors becomes even more complicated in Extract-Transform-Load (ETL) procedures where the data is heavily manipulated and/or aggregated from different sources. A quintessential ETL example is the DBpedia system, which implements sophisticated ETL procedures for extraction and refinement of data from semi-structured mixed-quality and crowd-sourced sources such as Wikipedia and Wikidata. Over the years, a huge community of users and contributors has formed around DBpedia, which are reporting errors via different communication channels such as Slack, Github and the DBpedia forum. A vast majority of the issues are associated with i) a piece of data and ii) a procedure (i.e. code) which has generated the data. While in the past the management of issues was done in an ad-hoc manner recently we introduced a systematic, test-driven approach for management of data and code related issues using Linked Data. In order to enable more efficient traceability and management of issues, we have introduced two technical improvements:

1. **Explicit association of data artifacts and code.** Previously DBpedia was grouped by language, which made backtracking difficult. Now every created and published data artifact is explicitly associated with the procedure (i.e. code) which created the artifact. For example, the “instance-types” artifact is associated with the “MappingExtractor.scala” class which created the artifact (“View code” action). This allows easier tracing of errors and relates data to code. A query on http://databus.dbpedia.org/sparql revealed that 26 code references exist and 12 are still missing for the wikidata group.

2. **Semantic pinpointing for issue management.** A major difficulty for tackling data issues was to identify in which file and version the error occurred. Team-internal discussions as well as submitted community issues did not have the proper vocabulary to describe the datasets, exactly. Using Databus identifiers, these errors can be pinpointed to the exact artifact, version and file.

3. **Test-driven approach for issue management (MiniDump).** Testing was mostly done after publishing (post-release) and reported issue were often ignored as reproduction of the error were either untraceable or required a full extraction (weeks) and difficult manual intervention. We created a test suite library that can be executed post-release as well as on a small-scale, extendable Wikipedia XML dump samples (collection of Wikipedia pages), producing a small release, i.e. a minidump. Tests on this minidump are executed on git push via continuous integration (minutes), thus enabling the following workflow: 1. for each reported data issue, a representative entity is chosen and added to the minidump. 2. a specific test at the appropriate level (see next section) is devised. 3. the code is improved so that the test passes. 4. post-release the same test is executed to check whether the fix was successful at larger scale, also testing for side-effects or breaking other parts of the software.

---

13 https://databus.dbpedia.org/dbpedia/mappings/instance-types/2020.04.01
14 https://git.informatik.uni-leipzig.de/dbpedia-assoc/marvin-config/-/tree/master/paper-supplement/codelink
The New DBpedia Release Cycle

Table 2. Testing methodology levels.

| Level       | Method     | Description                                                                 |
|-------------|------------|-----------------------------------------------------------------------------|
| Software    | JUnit      | Functional software tests on data parsers and extractor methods.            |
| Constructs  | Custom rules| IRI patterns and encoding errors, datatype and literal conformity and vocabulary usage. |
| Syntax      | Syntax parsing | Syntax parsing of output files implemented with Jena with customized selection of applicable errors and warnings. |
| Shapes      | SHACL      | A mix of auto-generated and custom SHACL test suites for domain and value range, cardinality and graph structure. |
| Integration | SPARQL over metadata | Verifies completeness of the releases and overall changes of quality metrics using Databus file/package metadata. |
| Consumer    | SPARQL on graph | Use case and domain specific SPARQL queries at consumer side. Point-of-truth evaluation. |

5 Testing Methodology

In order to cover the overall DBpedia knowledge management life cycle, from software development and debugging to release quality checks, we implemented robust “Testing Methodology” along six different levels listed in Table 2. The first level affects software development only. The next three levels (Constructs, Syntax and Shapes) are executed on the minidump as well as large scale. Integration and consumer (out of scope of this paper) are post-release only. In comparison, only software and post-release syntax and shapes were available before. The continuously updated developer wiki\(^{15}\) explains in detail, which steps are necessary to 1. add construct and SHACL tests, 2. extend the minidumps with entities, 3. configure the Jena parser and 4. run the tests and find related code. Besides the improvement in efficiency, the levels of testing were extended to cope with the variety of issues submitted to the DBpedia Issue tracker\(^{16}\).

5.1 Construct Validation

To investigate the layout and encoding conformity of produced data, we introduce an approach that concentrates on the in-depth validation of its pre-syntactical constructs. This concept differs from syntactical validation, since it does not rely on the complete syntactical correctness of the analyzed data, but checks the conformity for its single construct parts. A construct can be any character or byte sequence inside a data serialization, typically a specific part in the EBNF grammar. Moreover, a single construct can be validated independently of

\(^{15}\) [http://dev.dbpedia.org/Improve_DBpedia](http://dev.dbpedia.org/Improve_DBpedia)

\(^{16}\) [https://github.com/dbpedia/extraction-framework/issues?q=is%3Aissue+is%3Aopen+label%3Aci-tests](https://github.com/dbpedia/extraction-framework/issues?q=is%3Aissue+is%3Aopen+label%3Aci-tests)
inaccuracies in the rest of the data. This method can be used to gain better test coverage metrics over specific data parts, such as IRI patterns in RDF.

Assessing layout quality of an IRI is motivated by:

1. Linked Data HTTP requests are more lenient towards variation. RDF and SPARQL are strict and require exact match. Especially it is relevant that each release uses the exact same IRIs as before, which is normally not handled in syntactical parsing.

2. optional percent-encoding, especially for international chars and gen/sub-delims\(^{17}\) = ‘!’, ‘$’, ‘&’, ‘’’, ‘(’, ‘)’, ‘*’, ‘+’, ‘,’ ‘;’, ‘=’

3. Valid IRIs with the wrong name space http://www.wikidata.org/entity/Q64 or https://www.wikidata.org/wiki/Q64 or wrong layout (e.g. wkd:QQ64)

4. Correct use of vocabulary and correct linking

Complementary to syntactical validation, this approach provides a more fine-grained quality assessment methodology and can be specified as follows:

**Construct Test Trigger:** A construct trigger describes a pattern (e.g., a regular expression or wildcard) that covers groups of constructs (i.e. namespaces for IRIs) and assigns them to several domain-specific test cases. Moreover, if a trigger matches a given construct, then it triggers several validation methods that were assigned by a test generator. It is possible to define overlapping triggers, concerning that these patterns are highly flexible.

**Construct Validator:** To verify a group of triggered constructs, a test validator describes a specific reusable test approach. Therefore, several conformity constraints are currently implemented: *regex* - regular expression matching, *oneOf* - matching a static string, *oneOfVocab* - is contained in the ontology or vocabulary, and *doesNotContain* - does not contain a specific sequence. Further, we implemented generic RDF validators, based on Apache Jena, to test the syntactical correctness of single IRIs and literals.

**Construct Test Generator:** A construct test generator defines an 1 : n relation between a test trigger and several construct validators to describe a set of test cases. In the case of RDF NTriples, interesting constructs are IRIs or literals represented by the subject, predicate, or object part of a single triple. Blank

---

\(^{17}\) https://tools.ietf.org/html/rfc3987
nodes are ignored as they follow unpredictable patterns. For our approach, it was convenient to use Apache Spark and line-based regular expressions on NTriples to fetch these specific constructs. Listing 1 outlines an example construct test case specification covering DBpedia ontology IRIs, by checking the correct use of defined \texttt{owl:Class}, \texttt{owl:Datatype}, and \texttt{owl:ObjectProperties}. The construct validation approach seems theoretically extensible to validate namespaces, identifiers, attributes, layouts and encodings in other data formats like XML, CSV, JSON as well. However, we had no proper use case to justify the effort to explore it.

5.2 Syntactical Validation

The procedure of syntactical validation verifies the conformity of a serialized data format with its defined grammar. Normally, RDF parsers distinguish between different levels of "syntactical correctness", including errors and warnings. Errors represent entirely fraudulent statements, in the sense of irreproducible information, and a warning refers to an incorrect format of e.g., a datatype literal.

It is important to validate and clean the produced output of the DIEF, since some of the used methods are bloated, deprecated and erroneous. Therefore, the used syntax validation is configured to remove all statements containing warnings or errors. This guarantees better interoperability in the target software, which might use parsers considering some warnings as errors. The parser is a wrapper around Apache Jena, highly parallelized and is configured as fault-tolerant to skip erroneous triples and log exceptions correctly. The syntax cleaning process produces strictly valid RDF NTriples, on the one hand, and generates RDF syntax error reports, on the other. The original file is also kept on MARVIN to allow later inspection. The error reports provide structured input for community-driven and automated feedback. Finally, the valid NTriples are sorted to remove duplicated statements. This can later be utilized to compare iterations or modified versions of specific data releases.

5.3 Shape Validation
SAMPLE SPARQL integration test comparing file counts.

SHACL (Shapes Constraint Language)\(^{18}\) is a W3C Recommendation which defines a language for validating RDF graphs against a set of conditions. These conditions are provided as shapes and other constructs expressed in the form of an RDF graph. SHACL is used within the DBpedia’s knowledge extraction and release process to validate and evaluate the results (i.e. generated RDF). The defined SHACL tests are executed against the minidumps (Section 4.2).

Motivating example. Recently, the Czech DBpedia community has identified that the disambiguation links have not been extracted for Czech. The lack was discovered by an application-specific integration test (next section). Upon fixing the problem (configuration-related), a SHACL test (Listing 5.3) was implemented which will in future detect non-existence of the “disambiguation links” dataset on commit by checking a representative triple.

5.4 Integration Validation

Since software and artifacts possess a high coherence and loose coupling, additional methods are necessary to ensure overall quality control. To validate the completeness of a final DBpedia release, we use SPARQL queries on the produced Databus metadata. Listing 3 shows an example query to acquire an overview of the completeness of the mappings group releases on the DBpedia Databus\(^{19}\). Other application-specific tests exists, e.g. DBpedia Spotlight needs 3 files to compute a language model\(^{20}\).

---

\(^{18}\) edited by D. Kontokostas, the former CTO of DBpedia: https://www.w3.org/TR/shacl/

\(^{19}\) all groups https://git.informatik.uni-leipzig.de/dbpedia-assoc/marvin-config/-/tree/master/test

\(^{20}\) SPARQL query at https://forum.dbpedia.org/t/tasks-for-volunteers/163/Languages-with-missing-redirects/disambiguations
6 Experimental Evaluation

Section 3 and Table 1 has already introduced and discussed the size of the new releases. For our experiments, we used the versions listed there and in addition the MARVIN pre-release.

As a variety of methods (e.g. [4], a pre-cursor of SHACL) have been evaluated on DBpedia before and is not repeated here. We focused this evaluation on the novel construct validation, which introduce a whole previously invisible error class. Results are summarized, detailed reports will be linked to the Databus artifacts in the future. For this paper they are archived here\(^\text{21}\)

6.1 Construct Validation

To validate the constructs of the produced triples by DIEF, we specified generic and custom domain-specific test cases. With respect to the constructs in Section 5.1, we provide different test cases for IRI compliance and literal conformity to increase the test coverage over the extracted data. The IRI test cases focus on the encoding or layout of an IRI, and check the correct use of several vocabularies. In case of extracted DBpedia instance IRIs, the test cases validate the correctness considering that a DBpedia resource IRIs should not include sequences of '?' , '#', '[', ']' , '%21', '%24', '%26', '%27', '%28', '%29', '%2A', '%2B', '%2C', '%3B', '%3D' inside the segment part and follows Wikipedia conventions. The vocabulary test cases, which will be automated later, include tests for these schemas\(^\text{22}\): dbo, foaf, geo, rdf, rdfs, xsd, itsrdf, and skos to ensure the use of the respective ontology or vocabulary specification. Further, generic IRI and literal test cases are implemented to test their syntactical correctness and to validate the lexical format of typed literals. The full collection of specified custom construct validation test cases is versioned at the DIEF git repository\(^\text{23}\).

Construct Validation Metrics. We define construct validation metrics to measure the error rate and the overall test coverage for IRI patterns, encoding errors, datatype formats and vocabularies used in the produced data. The overall construct test coverage is defined by dividing the number of constructs that at least trigger one test by the total amount of found constructs.

\[
\text{Coverage} := \frac{\text{Triggered Constructs}}{\text{Total Constructs}}
\]

The overall error rate (in percent) is determined by dividing the number of constructs that have at least one error by the total number of covered constructs.

\[
\text{Error Rate} := \frac{\text{Erroneous Constructs}}{\text{Covered Constructs}}
\]

Test Results. The custom tests for the DBpedia generic and mappings release have an average of 87% IRI coverage (cf. Table 3). The test coverage can be increased by writing more custom test cases, but concerning the 80/20 rule, this could result in high efforts and the missing IRI patterns are presumably used inside of homepage or external link relations. The syntax cleaning was

\(^{21}\) https://git.informatik.uni-leipzig.de/dbpedia-assoc/marvin-config/-/tree/master/paper-supplement/reports
\(^{22}\) http://prefix.cc
\(^{23}\) https://github.com/dbpedia/extraction-framework/blob/master/dump/src/test/resources/dbpedia-specific-ci-tests.ttl
Table 3. Custom construct validation test statistics of the DBpedia and MARVIN release for the generic and mappings group (Gr). Displaying the total IRI counts, the construct validation test coverage, and construct errors of certain Databus releases.

| Gr.       | Release     | Version     | IRIs Total    | Coverage | Errors    | Error Rate |
|-----------|-------------|-------------|---------------|----------|-----------|------------|
| generic   | DBpedia     | 2016.10.01  | 12,228,978,594 | 83.93%   | 15,823,204 | 0.15%      |
|           | MARVIN      | 2019.08.30  | 11,089,492,791 | 90.98%   | 18,113,408 | 0.18%      |
|           | DBpedia     | 2019.08.30  | 11,089,492,791 | 90.98%   | 18,113,408 | 0.18%      |
|           | MARVIN      | 2020.04.01  | 10,527,299,298 | 89.59%   | 18,662,921 | 0.20%      |
|           | DBpedia     | 2020.04.01  | 10,022,095,645 | 89.32%   | 18,652,958 | 0.21%      |
| mappings  | DBpedia     | 2016.10.01  | 2,058,288,765  | 84.01%   | 6,692,902  | 0.39%      |
|           | MARVIN      | 2019.08.30  | 2,686,427,646  | 85.99%   | 6,951,976  | 0.30%      |
|           | DBpedia     | 2019.08.30  | 2,678,475,356  | 86.01%   | 6,875,930  | 0.30%      |
|           | MARVIN      | 2020.04.01  | 3,020,660,756  | 86.24%   | 7,514,376  | 0.29%      |
|           | DBpedia     | 2020.04.01  | 3,019,942,481  | 86.24%   | 7,505,332  | 0.29%      |

introduced on the '2019.08.30' version of the mappings release and later applied to the 'generic' release. It removes a significant amount of IRIs from the 'generic' version (~500 million) and only a fraction from the 'mappings' release, reflecting in the different extraction quality of them both. Although strict parsing was used and invalid triples removed, the other errors remain, which we consider a good indicator that the construct validation is complementary to syntax parsing.

Table 4 shows four independent construct validation test cases.

**XSD date literal (xdt):** This generic triple test validates the correct format use of xsd:date typed literals ("yyyy-mm-dd"^^xsd:date). Due to the use of strict syntax cleaning, as shown in Table 4, subsequent release later than 2016-10 do not contain incorrectly formatted date type literals, loosing several million. Removing warnings leads to better interoperability later.

**RDF language string (lang):** The DIEF uses particular serialization methods to create triples. These functions are often duplicated and contain deprecated code fragments. The post-processing module had an issue to build incorrect rdf:langString serializations by adding this IRI as explicit datatype instead of the language tag. Considering the N-Triples specification, this is an implicit literal datatype assigned by their language tags. This bug was not recognized by later parsers (i.e. Apache Jena), because the produced statements are syntactically correct. Therefore, to cover this behavior we introduce a generic test case for this kind of literals. The prevalence of this test is described by the pattern "*"^^rdf:langString" and the test validation is defined by an assertion that the pattern should not exist. Moreover, if a construct can be tested, the test directly fails and so the prevalence of the test is equal to its errors. A post-processing bug fix was provided before the 2020.04.01 release, and considering Table 4 was solved properly.

**DBpedia Ontology URIs (dbo):** To cover correct use of correct vocabularies,
Table 4. Construct validation results of the four test cases: XSD date literal (xdt), RDF language string (lang), DBpedia ontology (dbo) and DBpedia Instance (dbr?). We mention the total number of triggered constructs (prevalence), the aggregated amount of errors, and the percentile error rate for each test case.

| Gr. Test | Version | Prevalence | Errors | Error Rate |
|----------|---------|------------|--------|------------|
| DBp. generic | 2016.10.01 | 32,104,814 | 4,419,311 | 13.77% |
| | 2019.08.30 | 28,193,951 | 0 | 0% |
| | 2020.04.01 | 26,193,711 | 0 | 0% |
| DBp. mappings | 2016.10.01 | 229,009,107 | 229,009,107 | 100% |
| | 2019.08.30 | 353,220,047 | 353,220,047 | 100% |
| | 2020.04.01 | 0 | 0 | 0% |
| DBp. Instance | 2016.10.01 | 853,927,831 | 0 | 0% |
| | 2019.08.30 | 1,198,382,078 | 15,407 | 0.001286% |
| | 2020.04.01 | 1,354,209,107 | 0 | 0% |

some ontology test cases are specified. For the DBpedia ontology this test is assigned to the 'http://dbpedia.org/ontology/*' namespace and checks for correctly used IRIs of the DBpedia ontology. The test demonstrates that the used DBpedia ontology instances used inside the three ‘mappings’ release versions do not conform with the DBpedia ontology (cf. Table 4). By inspecting this in detail, we discovered the intensive production of a non-defined class `dbo:Location`, which is pending to be fixed. Error rate is lower in later releases, as size increased.

**DBpedia Instance URIs (dbr):** This test case checks the correct encoding of extracted DBpedia resource IRIs. Therefore, if a construct matches `http://[a-zA-Z]*.dbpedia.org/resource/*` the last path segment is checked to not contain the ‘?’ symbol as this kind of IRIs should never contain a query part. As displayed in Table 4 the incorrect extraction of the dbr IRIs considering the ‘?’ symbol occurred for version 2019.08.30 and was then solved in later releases.

### 6.2 Construct validation test coverage of non-DBpedia datasets

To show the re-usability of the construct validation approach, we analyzed a set of external RDF datasets²⁴. For these datasets our custom test cases achieved an average coverage around 10%. (cf. Table 5). The biggest part is covered by the custom vocabulary tests, especially foaf, rdf, rdfs and skos are commonly used across multiple RDF datasets. Another useful test case represents the correct use of DBpedia IRIs inside these datasets (inbound links). Almost in all external datasets, it could be recognized that backlinked DBpedia instances or ontology IRIs are wrong encoded or incorrectly used. In the case of RDF, this

²⁴ [https://databus.dbpedia.org/vehnem/collections/construct-validation-input](https://databus.dbpedia.org/vehnem/collections/construct-validation-input)
demonstrates that the introduced test approach can validate links between independently produced Linked Open datasets.

7 Related Work

An extensive base of our work is accommodated by [3], and the new DBpedia release cycle is built upon a broad set of these foundations. As the authors in [6] state, the DIEF persists to a number of strict limitations and needs to fulfill new substantial requirements to achieve a better and more sustainable development process. Therefore, we were inspired to introduce the minidump tests by offering an extensible and community-driven concept.

Data Release Cycle. The release processes for different knowledge bases are naturally different due to the different ways of obtaining the data. Considering Wikimedia as a source of the DBpedia extraction, the Wikidata project releases dumps on a weekly basis and publishes them in an online file directory. The 2020-03-30 release provides 1 billion statements, while the 2020-04-01 DBpedia release has 13.5 billion triple statements. Moreover, DBpedia provides machine-readable descriptions for all published data artifacts, which are also indexed and findable via the DBpedia Databus. Due to the different nature, DBpedia implements software/minidump and large-scale validation mechanism. Wikidata uses ShEx on user input. Besides Wikimedia, there are other open data releases, like WordNet, FrameNet, and Yago, to name a few. However, their processes are slightly intransparent and most of them do not focus on time-based releases.

Data Quality Assessment. Further, we briefly mention two projects that attempt Linked Data quality assessment by applying alternative facets. TripleCheck-Mate [1] describes a crowd-sourced data quality assessment approach by producing manual error reports of whether a statement conforms to a resource or can be classified as a taxonomy-based vulnerability. Their results showed a broad overview of examined errors but were tied to high efforts and offered no integration concept for further fixing procedures. On the other hand, RDFUnit is a test-driven data-debugging framework that can run automatically generated and manually generated test cases (predecessor of SHACL) against RDF datasets [4].
These automatic test cases mostly concentrate on the schema, whether domain types, range values, or datatypes adhere correctly. The results are also provided in the form of aggregated test reports.

8 Conclusion and Future Work

In this paper, we presented an agile and efficient knowledge extraction workflow in the case of the novel DBpedia release. Therefore, we primarily focus now on time-based, test-driven and traceable data development aspects. Considering that DBpedia is an enormous open source project, we introduced a new set of extensive test methods, to offer a convenient process for community-driven feedback and development. The DBpedia Databus was appointed as a quality controlled interface, due to the utilization of traceable metadata. The construct validation test approach provides a more in-depth issue tracking checking for wrong formatted datatypes, inconsistent use of vocabularies, and the layout or encoding of IRIs produced in the extracted data. In combination with syntactical and shape validation, this covers a large spectrum of possible data flaws. Moreover, it was shown that the minidump-based and large-scale test concept provides a flexible view to directly link tests with existing issues. The described workflow builds a reliable and stable base for future DBpedia (or other quality-assured data) releases, but the complete effectiveness can only be measured in the coming years. As an overall result, the new DBpedia release cycle produces on average over 13.5 billion triples per month with a minimal publishing effort. For future upcoming releases, user interface will be created and all evaluation reports will be linked to Databus artifacts similar to the explained code references.

References

1. Acosta, M., Zaveri, A., Simperl, E., Kontokostas, D., Auer, S., Lehmann, J.: Crowdsourcing linked data quality assessment. In: ISWC. pp. 260–276. Springer (2013)
2. Beck, K., Beedle, M., van Bennekum, A., et al.: Manifesto for agile software development (2001), http://www.agilemanifesto.org/
3. Feeney, K., Davies, J., Hellmann, S.: Engineering Agile Big-Data Systems. River Publishers (2018)
4. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A.: Test-driven evaluation of linked data quality. In: WWW (2014)
5. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. SWJ 6(2) (2015)
6. Maroy, W., Dimou, A., Kontokostas, D., Meester, B.D., Lehmann, J., Mannens, E., Hellmann, S.: Sustainable linked data generation: The case of dbpedia. In: ISWC (2017)
7. Samuelson, P.A., Nordhaus, W.D.: Microeconomics /. McGraw-Hill Irwin., Boston:, 17th ed. edn. (2001)
8. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment for Linked Data: A survey. Semantic Web Journal (2015)