A Two-Fold Quality Assurance Approach for Dynamic Knowledge Bases: The 3cixty Use Case

Nandana Mihindukulasooriya[1], Giuseppe Rizzo[2], Raphaël Troncy[3], Oscar Corcho[1], and Raúl García-Castro[1]

[1] Ontology Engineering Group, UPM, Spain,
{nmihindu, ocorcho, rgarcia}@fi.upm.es
[2] ISMB, Italy,
giuseppe.rizzo@ismb.it
[3] EURECOM, France,
raphael.troncy@eurecom.fr

Abstract. The 3cixty platform relies on a continuous integration workflow for the generation and maintenance of evolving knowledge bases in the domain of culture and tourism. This approach is inspired by common practices in the software engineering industry in which continuous integration is widely-used for quality assurance purposes. The objective of this paper is to present a similar approach for knowledge base population and publishing. The proposed approach consists of two main steps: (i) exploratory testing, and (ii) fine-grained analysis. In the exploratory testing step, the knowledge base is tested for patterns that may reveal erroneous data or outliers that could indicate inaccuracies. This phase is knowledge-base agnostic and provides inputs for the second phase. In the fine-grained analysis step, specific tests are developed for a particular knowledge base according to the data model and pre-defined constraints that shape the data. More precisely, a set of predefined queries are executed and their results are compared to the expected answers (similar to unit testing in software engineering) in order to automatically validate that the knowledge base fulfills a set of requirements. The main objective of this approach is to detect and to flag potential defects as early as possible in the data publishing process and to eliminate or minimize the undesirable outcomes in the applications that depend on the knowledge base, typically, user interfaces that enable to explore the data but rely on a particular shape of the data. This two-fold approach proves to be critical when the knowledge base is continuously evolving, not necessarily in a monotonic way, and when real-world applications highly depend on it such as the 3cixty multi-device application.

Keywords: Data quality, data validation, continuous integration, dynamic knowledge base

1 Introduction

Quality is an important aspect for both the development and maintenance of a knowledge base (KB) in the same way as it is to software engineering. A lot of work has been done on quality assurance in software engineering and a plethora of tools and technologies have been developed and used for validating the quality of software products. We

argue that the KB generation process should learn from the experiences in the software engineering field and adapt some of these approaches for assuring KB quality.

In modern days of complex software development, it is hard to envision the complete set of user requirements upfront and to immediately have all functionalities and data models fully-defined. Both functionalities and data models of the software should evolve based on user feedback, changes in business and technical environments, and other organizational factors [4]. Thus, software development methodologies that adapt to changes (rather than rejecting unforeseen changes) such as agile methodologies, became inevitable in the software development industry. However, despite of the benefits those methodologies bring, such as business agility or faster development and release cycles, one of the main challenges of these methodologies is how to perform quality assurance to validate that software meets the user requirements and does not introduce any undesirable side effects [16]. Continuous integration [5] plays a key role in guaranteeing the quality of systems that have components or data that evolve rapidly and are developed by multiple teams. Continuous integration allows teams to integrate their work frequently while using automated builds and testing mechanisms for detecting quality problems as early as possible thus reducing the cost of correcting those problems and build high quality functional software rapidly [9].

3cixty is a semantic web platform that enables to build real-world and comprehensive knowledge bases in the domain of culture and tourism for cities. The entire approach has been tested first for the occasion of the Expo Milano 2015 [18], where a specific knowledge base for the city of Milan was developed, and is now refined with the development of knowledge bases for the cities of Nice and London. They contain descriptions of events, places (sights and businesses), transportation facilities and social activities, collected from numerous static, near- and real-time local and global data providers, including Expo Milano 2015 official services in the case of Milan, and numerous social media platforms. When deploying 3cixty on a regular basis, we observe the imperial need of following an agile process. The knowledge base is continuously evolving, since new cultural events are proposed every day, while some other resources cease to exist, e.g., a business has moved or has closed or is changing its opening hours. The main motivation for the approach proposed in this paper is to validate the evolution of the content of a knowledge base while maintaining a high level of quality. Furthermore, it is also useful to monitor the growth of the knowledge base providing a high-level view of its evolution.

Data quality is a complex multi-dimensional concept commonly defined as “fitness for use” [20]. The data quality life-cycle generally includes the identification of quality requirements and relevant metrics, quality assessment, and quality improvement. The metrics for quality assessment are categorized in different dimensions such as consistency, conciseness, or completeness. For each of these dimensions, there are a wide-range of metrics defined in the literature for measuring different aspects related to a given quality dimension [21]. On the one hand, some of these metrics can be knowledge base agnostic such as the number of triples in the KB or the number of syntax errors in a particular serialization format of the KB. On the other hand, some other metrics are specific to the KB, for example, the number of bus stops that do not have geo coordinates in the 3cixty KB. These two types of metrics are complementary such that

more generic and KB agnostic metrics provide hints for defining fine-grained KB specific metrics. Quality assessment can be done either manually or (semi-)automatically with the help of the tools that facilitate the generation of the required metrics.

Once the quality assessment is done, the assessment results will indicate the presence of quality issues. The ultimate goal is to improve the overall KB quality based on the quality issues identified by eliminating them and making sure that they will not be introduced in the future. In this paper, we present the data quality approach we used for developing the 3cixty KB which uses a continuous integration approach inspired by the software engineering industry.

The remainder of this paper is structured as follows. We discuss some related work in Section 2. Then, we present a two-fold approach for quality assurance of dynamic KBs in Section 3. We detail how this approach has been implemented in the case of the 3cixty KB, using the LOUPE and SPARQL interceptor tools in Section 4. We propose some lessons learned in Section 5 and we finally outline some future work in Section 6.

2 Related Work

Related work on the topic of quality assessment of knowledge bases can be categorized into two main groups: a) research works that define techniques and metrics for measuring quality, and b) tools and applications that assess quality. Zaveri et al. [21] provide a comprehensive literature review of the efforts related to Linked Data quality evaluation, with a classification of quality dimensions and metrics found in the literature. This work includes 69 metrics grouped into 18 quality dimensions extracted from 30 research papers published between 2002 to 2014. More recently, Assaf et al. [1] build upon these efforts but focus only on objective quality indicators, and propose the ROOMBA tool that helps data owners to rate the quality of their datasets and get some hints on possible improvements, and data consumers to choose their data sources from a ranked set.

Quality assessment tools and applications can be categorized into two main groups. First, tools such as ProLOD [3], LODStats [6] and ABSTAT [17] that profile datasets and generate statistics which provide heuristics and general indicators about the dataset quality. In the approach presented in this paper, we use a similar tool, Loupe [15], for exploring the 3cixty KB and detecting quality issues. The main advantage of Loupe compared to the aforementioned tools is that it allows the users to deep dive into high-level statistics information by zooming into several levels of details, making it easier to identify outliers and abnormal patterns in the data. Furthermore, the detailed information presented in Loupe is directly linked to the corresponding triples via dynamically-generated SPARQL queries making it easier for the quality assessor to inspect the corresponding triples.

Other quality assessment tools enable to perform a more fine-grained analysis of knowledge bases. WIQA [2] allows to filter information based on policies defined by users in the WIQA-PL policy language. Sieve [14] is a framework that attempts to assess and to increase completeness, conciseness and consistency of data through data fusion. DaCura [7] provides a set of tools for collecting and curating evolving datasets maintaining high-quality Linked Data. Triplecheckmate [13] is a tool for crowdsourcing the quality assessment of Linked Data. RDFUnit [12] is a tool for test-driven quality

assessment based on schema constraint validation detection with SPARQL query templates and it is the closest to the SPARQL Interceptor system presented in this paper. SPARQL Interceptor has several characteristics that makes it particularly suitable for the approach presented in this paper. It is a tool well-thought for continuous integration and delivery, and it includes useful features such as continuous monitoring of the KB status, breakdown analysis per status of the KB, and configurable notifications. It seamlessly integrates with continuous integration systems such as Jenkins [4] and can be easily used in the knowledge base development process.

3 A two-fold approach for quality assurance of continuously evolving KBs

The Linked Data quality assurance methodologies generally consist of (i) identification of quality requirements, (ii) execution of quality assessment, and (iii) data repair and quality improvements [19]. As quality is defined as "fitness for use" and quality is multidimensional, it is important to identify what are the important aspects of quality dimensions that are relevant and what are the required levels of quality for a given use case. The quality assessment is planned according to the information extracted in the quality requirements identification phase. The quality assessment can be done at different levels. First, a generic analysis of the knowledge base can be done by extracting a set of common knowledge base agnostic metrics that can give certain heuristics about the data quality. Then, a more advanced analysis can be done which is focused on the quality requirements identified during the first phase to ensure that those requirements are fulfilled in the dataset. The final phase is the data repair and quality improvement step which has the goal of identifying the causes of the quality issues and of eliminating them. This is important because the dataset generation process could have several steps such as the acquisition of raw data, RDF mapping and transformations, data publication and each of these steps can introduce quality defects into data. Thus, the root cause of the quality defects needs to be identified and eliminated so that not only the current version of the dataset but also the future versions will be free of errors.

This section presents a methodology suitable for quality assessment of continuously evolving knowledge bases that are consumed by real-world applications. The main objective of the methodology is to ensure that the periodically generated datasets (e.g., on a daily basis) meet the quality requirements of the use cases. The methodology consists of two main phases: (i) exploratory testing of the KB using dataset statistics and common patterns, and (ii) fine-grained analysis of the KB aligned with the quality requirements.

3.1 Exploratory testing

Exploratory testing (also known as ad-hoc testing) is a branch of blackbox software testing where the testers simultaneously learn, design and execute tests in an exploratory manner [11]. In exploratory software testing, a tester would explore the software capabilities without a pre-defined test plan and design the tests on the fly based on the

[4] `https://jenkins-ci.org`

information gained while testing. This same idea, which is quite successful in the software industry [10], can be adapted to testing knowledge bases. In knowledge bases, exploratory testing can be performed by a tester by executing different queries on the knowledge base, analyzing them, and incrementally planning more queries based on the knowledge acquired. This requires the testers to write a lot of boiler-plate queries for extracting information about different aspects such as the usage of classes or properties of different vocabularies or other patterns in the knowledge base. This process can be accelerated and made more efficient using a set of query templates that extract statistics about the knowledge base or the patterns of vocabulary usage. These query templates can extract both statistics that are generic such as the number of triples in the knowledge base, the number of instances of a specific class (e.g., the number of bus stops or the number of business places that belong to the category `restaurant`). In addition to finding defective patterns, by observing the evolution of these values after each new deployment, one can monitor the growth of the knowledge base.

Testers can explore the statistics of the knowledge base and identify potential outliers. For example, all instances of a given class can be analyzed for checking what are the properties that are used with those instances and their cardinality. Concretely, a tester can analyze what is the cardinality of the geo location properties associated with the instances of the `dul:Place` class that belong to the `YelpBusiness` category. Whilst exploring the data instances, a tester might identify that there are certain individuals that have multiple geo location properties or some individuals without a geo location property even though the most common pattern is to have exactly one geo location property. This could indicate a quality problem in the knowledge base and the tester may report this to the developers to find the root cause of this inconsistency so that the developers correct the errors in the knowledge base generation process. Further, this indicates the need for including fine-grained analysis tests for this property so that if the same errors are repeated, they will be detected again. Thus, exploratory testing is complementary to fine-grained testing.

Furthermore, it is possible to analyze the subjects and objects of all the triples containing a given property, which enables to count the number of IRIs and blank nodes related by this property for all subjects, and to analyze the type of each subject. This information allows the exploratory tester to identify cases where the data does not fit a priori the schema (e.g., violation of domain and range constraints) or other concrete modelling decisions (e.g., no blank nodes used) that were made during the use case requirements phase.

Exploratory testing of the KB allows to identify incorrect values such as outliers. For example, if more than 99% of the values of the `schema:interactionCount` property are positive integers but one finds two negative values for this property, it is recommended to look into those two particular values. As discussed above, the goal of the first phase of the approach is to adapt exploratory testing for maintaining quality of the knowledge base generation process.

3.2 Fine-grained Analysis of a KB

The goal of the fine-grained analysis of the KB is to perform a more systematic analysis using a set of test queries that are KB-specific. These test queries are run automatically

every time a new version of the KB is generated and if any defects are found, they are reported to the developers. Fine-grained testing follows a similar approach to unit tests in software engineering in which tests are incorporated to the knowledge base through a continuous integration process.

Similar to the software engineering process, these tests are planned and designed in advance based on the constraints and validation rules of the KB and on the requirements of different use cases of the application that are built on top of the KB. These fine-grained test scripts can check various quality dimensions such as syntactic validity, consistency, completeness, or correctness. For instance, for syntactic validity, certain values such as the location URI are checked for well-formed URI strings or the values of date fields are checked for correct date formats with a data-type declaration.

Regarding consistency, the 3cixty data model requires mandatory properties for some entities. For example, all entities of type `dul:Place` or `lode:Event` must have geo-coordinates (using respectively the `geo:location` or `lode:inSpace` property). Thus, tests are included to verify that all instances of `dul:Place` or `lode:Event` have such a property. Similarly, other cardinality restrictions and functional properties can be converted into tests in the fine-grained analysis. For instance, all entities of type `dul:Place` or `lode:Event` must also have exactly one `lode:poster` property while all entities of type `lode:Event` must have at most one `dc:title` per language.

For completeness, the knowledge base can be checked to see whether it contains all the entities of a given type. For instance, the number of bus stops (or metro stations) in Milan is known. Therefore, a test can be included to check if the knowledge base contains information about all bus (metro) stops in Milan. However, it is important to update regularly such tests whenever those values change so the tests are synchronized with the context of the knowledge base.

Different facts in the knowledge base can be checked for correctness using different techniques. For example, the latitude (`geo:lat`) and longitude (`geo:long`) property values of places in a given city must belong to a particular range (greater than a lower bound and lesser than an upper bound) corresponding to the city bounding box area. Hence, for each city described in the knowledge base, tests can be included to identify the entities that are *a priori* located outside of the city grid.

4 Implementation: Loupe and SPARQL Interceptor

The two-fold methodology described in the previous section has been implemented using two tools: Loupe and SPARQL Interceptor.

4.1 Loupe

Loupe[5] is a tool that supports the inspection of datasets to understand which vocabularies (classes and properties) are used, including statistics and frequent triple patterns. Starting from high-level statistics, Loupe allows a tester to dig into details down to the

[5] `http://loupe.linkeddata.es`

corresponding triples and analyze potential quality problems. As illustrated in Figure 1, Loupe consists of two main components: (a) Loupe Core and (b) Loupe UI (User Interface). In order to make the knowledge-base inspection seamless and more efficient, Loupe performs an initial indexing of the statistics and the indexed statistics and patterns are stored in an Elasticsearch server. Loupe makes also use of a docker image of a Virtuoso server[6] to ease the indexing process, given an RDF dump, and to avoid overloading the public SPARQL endpoint of the knowledge base.

Once the knowledge base is indexed, the Loupe UI provides an intuitive web interface for exploratory testers to inspect the knowledge base. Loupe UI consumes the index generated by Loupe Core and generates several UI components including a summary, a class explorer, a property explorer, a triple pattern explorer, and an ontology explorer. Each of these UI components provides several levels of information starting from high-level overviews and then allowing users to navigate into further details. Loupe UI allows testers to identify the different defective patterns that were described in Section 3.1.

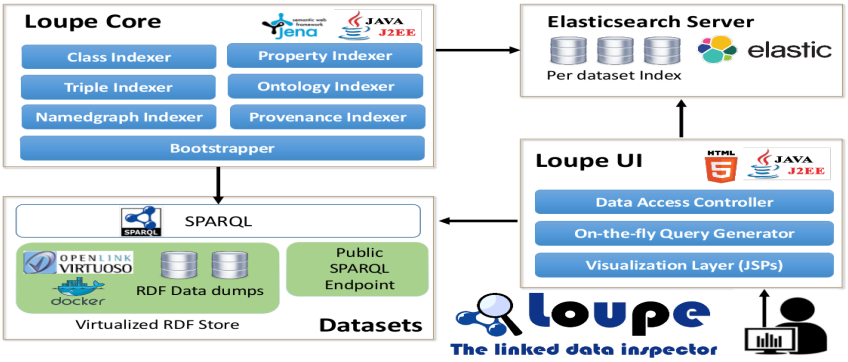

Fig. 1. Loupe: The Linked Data Inspector

4.2 SPARQL Interceptor

SPARQL Interceptor[7] integrates a set of user-defined SPARQL queries (namely unitary tests for an RDF dataset) inside the Jenkins continuous integration system. SPARQL queries are defined by the dataset developers, with the objective of checking whether some specific characteristics hold in the released dataset. For example, one may want to check that a property is systematically valued for a particular entity: in 3cixty, all entities of type `dul:Place` must have geo-coordinates. One may also want to check

[6] `https://hub.docker.com/r/nandana/virtuoso-oss-7-loupe/`
[7] A deployed instance is available at `http://alzir.dia.fi.upm.es:8080`

that all entities are encompassed in a particular geographical coverage: in 3cixty, a city is delimited by a rectangular bounding box area and all entities of type `dul:Place` must be geolocalized within this bounding box. These queries are evaluated regularly, something that is done automatically by Jenkins, and summary reports are provided on the compliance of the outputs of these query evaluations with the expected results as described in the definition of the queries. Queries can then check both schema inconsistencies and data instances. The Figure 2 shows a snapshot of the SPARQL Interceptor in a real setting. Error logging follows the conventional presentation layout provided in

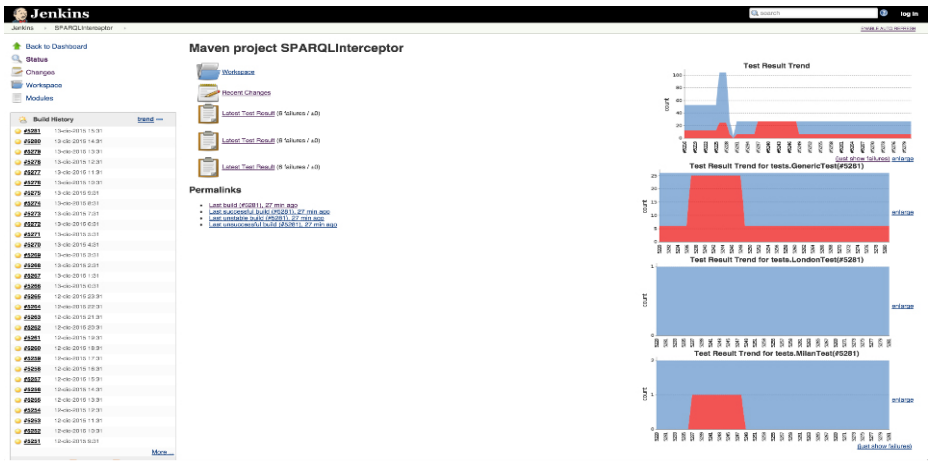

Fig. 2. Screenshot of SPARQL Interceptor: on the left hand side, we observe the build history; the center groups set of results of the run of the system; on the right hand side, the charts provide an overview of the different status of the runs.

Jenkins with a general status of the build and a breakdown per test. For each of the failed tests, the system provides the reason for the failure, plus a description of the query and a link to execute it against the SPARQL endpoint, which is useful for the expert while checking the obtained results.

5 Discussion and Lessons Learned

We use both Loupe and SPARQL Interceptor over six months to monitor the continuous deployment of the evolving 3cixty knowledge base which was at the same time used by a real application available to the million potential visitors of EXPO 2015. In this section, we report on some lessons learned, either frequent errors that we observe over iterative deployments of the knowledge base, or additional functionalities that Loupe and SPARQL Interceptor could provide.

One of the challenges that the testers have, when using Loupe for exploratory testing, is to find the initial set of uncommon patterns so that they can explore more to find potential defects. This happens specially when the knowledge base under test is

significantly large (both in the amount of facts as well as the number of different vocabularies used) and when the tester does not have sufficient domain knowledge about the knowledge base. We consider two different approaches for addressing this challenge. On the one hand, Loupe can provide reports that highlight the uncommon patterns and potential outliers. These reports can use different outlier detection techniques such as population distribution based approaches or Local Outlier Factor (LOF) [8]. We envision that these reports will significantly reduce the amount of time spent to find the potential outliers and allowing more time to analyze them and dig into more details. The testers can identify the possible natural outliers (unusual values or patterns that are indeed correct values, i.e., false positives) using external information resources about the city and only analyze and report the erroneous outliers. This feature is planned as future work in the roadmap of Loupe. Alternatively, another approach requested by some testers to handle the same problem is to provide the ability to export the knowledge base statistics in a data exchange format such as CSV so that they can analyze some suspecting value populations or patterns in a tool of their choice such as R or Excel. This allows testers to use some of the tools and techniques that they are familiar with to analyze the data.

One challenge in designing fine-grained tests is to keep them synchronized with the external data sources being used to build the knowledge base. Some tests, such as the ones related to cardinality of the data model, are sometimes precise and include exact values, while others are values greater or lesser than a certain threshold. For instance, if we consider the number of bikes available in a given place, such a value changes over time, therefore setting up a priori threshold may not be effective or it would require a certain degree of abstraction that a highly dynamic context would not support. On the expert side, a plethora of error messages is something that we avoid, thus constraining the fine-grained test query types to checking to:

presence of unexpected properties or property values Evaluate against a Boolean value. In 3cixty, we are interested in points of interest that have an added value for the users and we delete in particular all places that are categorized as residential places that we have harvested during the scraping process. Therefore, we check using a unit test that no more ”home places” are present in the KB once deployed;

presence of properties Evaluate against a Boolean value. In 3cixty, both Place-type and Event-type instances have required properties so we check whether there are instances with missing values for those required properties or not;

presence of duplicated properties Evaluate against a Boolean value. In 3cixty, some properties cannot have more than one value so we check whether there are duplicates in the property set or not;

syntax Evaluate against a Boolean value. These tests check the syntax of the data value;

presence of unsolicited instances Evaluate against a Boolean value. These tests check whether an instance is present despite it should not, for example, Place-type instances that do not belong to any cell contained in the grid;

number of graphs Evaluate against a cardinal number. In 3cixty, data is always associated to a publisher graph which enables to check that proper attribution has been made;

number of instances/properties Evaluate against a cardinal number. These tests enable to check that the right number of properties is being used in a particular 3cixty deployment.

This classification offers a general picture of the types of query that are been tested in the 3cixty platform. Those benchmark queries are classified into: *i)* generic test queries and *ii)* city-related test queries (such as the ones evaluating the number of instances/properties that depend on the final numbers we can observe in the given city). SPARQL Interceptor visualizes such a classification in the UI, grouping the queries according to the domain and the city (namely Milan, London, Nice in the case of 3cixty). Despite that this was necessary in analysing the KB and the different components, this has increased the complexity for the KB developers in generalizing as much as possible the queries and, thus, evaluating them.

6 Conclusions and Future Work

This paper presents a two-fold approach for ensuring the quality of dynamic knowledge bases that evolve continuously. This approach is inspired by several popular techniques in software engineering such as exploratory testing and continuous integration. Based on our experiences in the 3cixty use case, we conclude that these practices of software engineering can be adapted to knowledge-base generation. They help to maintain the quality of a KB while allowing its rapid evolution. We advocate a two-phase approach enabling to provide an overview of the evolution of a KB and a fine-grained analysis. We have described Loupe and SPARQL Interceptor, two tools that provide the necessary features for performing this two-phase approach.

Future work for Loupe includes improvements in (a) generation of quality reports, (b) utilization of machine learning techniques for (semi)-automatic detection of outliers and potentially defective patterns, (c) improvements to the visualizations. About the fine-grained analysis, future research activities will be carried out to explore the automatic generation of KB-specific test queries. This will save time in the generation ad-hoc queries, and this will attract the attention of the expert or developers in updating the queries and correctly checking the number of instances or properties being recently deployed. We will also consider expressing the SPARQL Interceptor test queries using the new Shapes Constraint Language (SHACL[8]) being standardized by W3C.

Acknowledgments

This work was partially supported by the FPI grant (BES-2014-068449), the innovation activity 3cixty (14523) of EIT Digital, and the 4V project (TIN2013-46238-C4-2-R).

References

1. Assaf, A., Troncy, R., Senart, A.: An Objective Assessment Framework & Tool for Linked Data Quality: Enriching Dataset Profiles with Quality Indicators. International Journal on Semantic Web and Information Systems (IJSWIS) (2016)

[8] `https://www.w3.org/TR/shacl/`

2. Bizer, C., Cyganiak, R.: Quality-driven information filtering using the WIQA policy framework. Web Semantics: Science, Services and Agents on the World Wide Web 7(1), 1–10 (2009)
3. Böhm, C., Naumann, F., Abedjan, Z., Fenz, D., Grütze, T., Hefenbrock, D., Pohl, M., Sonnabend, D.: Profiling linked open data with ProLOD. In: 26^{th} IEEE International Data Engineering Workshops (ICDEW). pp. 175–178 (2010)
4. Cockburn, A., Williams, L.: Agile software development: it's about feedback and change. Computer 36(6), 39–43 (2003)
5. Duvall, P., Matyas, S., Glover, A.: Continuous integration: improving software quality and reducing risk. Pearson Education (2007)
6. Ermilov, I., Martin, M., Lehmann, J., Auer, S.: Linked open data statistics: Collection and exploitation. In: 4^{th} International Conference on Knowledge Engineering and the Semantic Web (KESW). pp. 242–249 (2013)
7. Feeney, K.C., O'Sullivan, D., Tai, W., Brennan, R.: Improving curated web-data quality with structured harvesting and assessment. International Journal on Semantic Web and Information Systems (IJSWIS) 10(2), 35–62 (2014)
8. Fleischhacker, D., Paulheim, H., Bryl, V., Völker, J., Bizer, C.: Detecting errors in numerical linked data using cross-checked outlier detection. In: The Semantic Web–ISWC 2014, pp. 357–372. Springer (2014)
9. Fowler, M., Foemmel, M.: Continuous integration. Thought-Works (2006)
10. Itkonen, J., Mäntylä, M.V., Lassenius, C.: Defect detection efficiency: Test case based vs. exploratory testing. In: Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. pp. 61–70. IEEE (2007)
11. Itkonen, J., Rautiainen, K.: Exploratory testing: a multiple case study. In: International Symposium on Empirical Software Engineering (2005)
12. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A.: Test-driven Evaluation of Linked Data Quality. In: 23^{rd} World Wide Web Conference (WWW). pp. 747–758. ACM (2014)
13. Kontokostas, D., Zaveri, A., Auer, S., Lehmann, J.: Triplecheckmate: A tool for crowdsourcing the quality assessment of linked data. In: 4^{th} International Conference on Knowledge Engineering and the Semantic Web (KESW). pp. 265–272 (2013)
14. Mendes, P.N., Mühleisen, H., Bizer, C.: Sieve: linked data quality assessment and fusion. In: 15^{th} International Joint EDBT/ICDT Workshops. pp. 116–123 (2012)
15. Mihindukulasooriya, N., Villalon, M.P., García-Castro, R., Gomez-Perez, A.: Loupe-An Online Tool for Inspecting Datasets in the Linked Data Cloud. In: 14^{th} International Semantic Web Conference (ISWC), Posters & Demonstrations Track (2015)
16. Nerur, S., Mahapatra, R., Mangalaraj, G.: Challenges of migrating to agile methodologies. Communications of the ACM 48(5), 72–78 (2005)
17. Palmonari, M., Rula, A., Porrini, R., Maurino, A., Spahiu, B., Ferme, V.: ABSTAT: Linked Data Summaries with ABstraction and STATistics. In: 12^{th} Extended Semantic Web Conference (ESWC), Posters & Demonstrations Track. pp. 128–132 (2015)
18. Rizzo, G., Troncy, R., Corcho, O., Jameson, A., Plu, J., Ballesteros Hermida, J., Assaf, A., Barbu, C., Spirescu, A., Kuhn, K., Celino, I., Agarwal, R., Nguyen, C., Pathak, A., Scanu, C., Valla, M., Haaker, T., Verga, E., Rossi, M., Redondo Garcia, J.: 3cixty@Expo Milano 2015: Enabling Visitors to Explore a Smart City. In: 14^{th} International Semantic Web Conference (ISWC), Semantic Web Challenge (2015)
19. Rula, A., Zaveri, A.: Methodology for Assessment of Linked Data Quality. In: Workshop on Linked Data Quality (2014)
20. Wang, R., Strong, D.: Beyond accuracy: What data quality means to data consumers. Journal of management information systems 12(4), 5–33 (1996)

21. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment for linked data: A survey. Semantic Web Journal 7(1), 63–93 (2015)