IFS INSTITUTE FOR SOFTWARE

OST Ostschweizer Fachhochschule

Qualitätsprüfung der Verknüpfungen von OpenStreetMap nach Wikidata

Bachelorarbeit
Studiengang Informatik

Timon Erhart
timon.erhart@ost.ch

Jari Elmer
jari.elmer@ost.ch

OST Ostschweizer Fachhochschule
Rapperswil

Betreuer:
Prof. Stefan Keller

Projektpartner:
Sascha Brawer

16. Juni 2022

Abstract

Die freie Landkarte OpenStreetMap ist in der Lage, Verknüpfungen auf die strukturierte Wissensdatenbank Wikidata abzubilden. Derzeit existieren rund 5.5 Millionen solcher Verknüpfungen von OpenStreetMap zu Wikidata. Die Qualität dieser Verknüpfungen ist bislang jedoch unbekannt und ungeprüft.

Die entwickelte Applikation `osm wikidata quality checker` überprüft diese Verknüpfungen auf deren Qualität. Sie läuft in einem Docker-Container, besorgt sich selbstständig die Datensätze, verarbeitet diese und liefert gefundene Fehler an die etablierte Fehlerdatenbank Osmose. Ziel dieser Arbeit ist, dass die Applikation ein dauerhafter Teil der Infrastruktur zur Qualitätssicherung von OpenStreetMap wird und einfach weiterentwickelt werden kann.

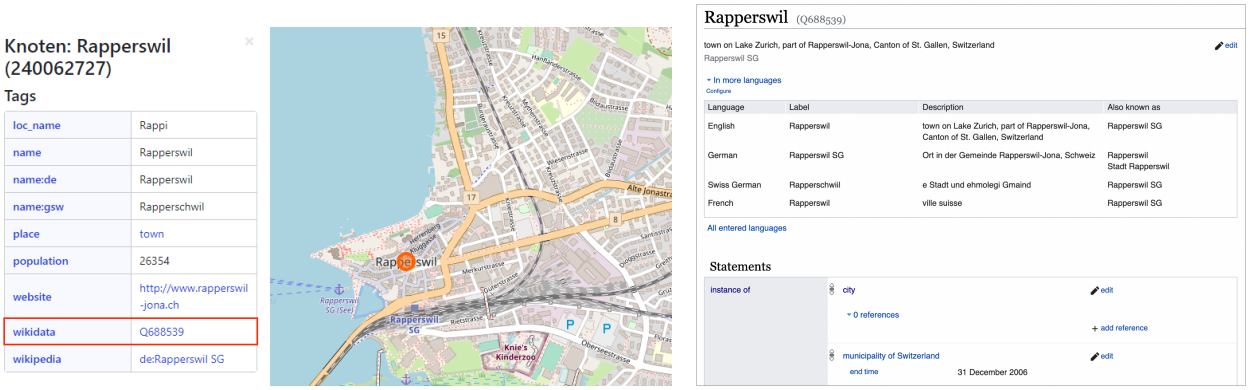
Die Grundlage für die Checks liefern die Datenbank-Dumps von OpenStreetMap und Wikidata. Die Herausforderung bestand darin, die enorme Datenmenge innerhalb einer Woche und mit geringem Speicherplatzbedarf zu verarbeiten. Dies wurde hauptsächlich durch den Einsatz von Multiprocessing und des entwickelten Datenbankmodells, bei dem nur die relevanten Daten extrahiert werden, erreicht. Auch Schwierigkeiten im Umgang mit crowdsourced Data, bei denen mit unvorhergesehene Datenfehlern gerechnet werden muss, wurden erfolgreich gemeistert, sodass eine hohe Fehlertoleranz erreicht wurde. Eine ausführliche Dokumentation sowie die leicht verständliche Architektur ermöglicht es, das Tool auszubauen und weitere Checks zu implementieren. Die optionale Konfiguration bietet die nötige Flexibilität im Betrieb und hilft bei der Weiterentwicklung.

Entstanden ist ein betriebsreifes Werkzeug, welches in der Lage ist, mit den grossen Datenmengen umzugehen und die gesamte Welt innerhalb der geforderten Frist zu prüfen. Aktuell werden weltweit über 30’000 Fehler in neun Kategorien gefunden. Dabei wird eine hohe Treffsicherheit von 95% erreicht.

Management Summary

Ausgangslage

OpenStreetMap ist eine freie Landkarte der gesamten Welt mit rund einer Milliarde geografischen Objekten. Wikidata ist eine freie Wissensdatenbank mit über 97 Millionen Einträgen, die strukturierte Daten zur Verfügung stellt. Beides sind crowdsourced Open-Data-Projekte mit grossen und aktiven Communities, welche die Daten laufend aktualisieren und ergänzen. Seit 2014 ist OpenStreetMap in der Lage, über sogenannte Tags (Key-Value-Pairs) eine Verknüpfung zu Wikidata herzustellen. Abbildung 1 zeigt eine solche Verknüpfung für den Ort Rapperswil. Zurzeit existieren rund 5.5 Millionen Wikidata-Tags mit einer stetig wachsenden Popularität. Mittels dieser Verknüpfung lassen sich interessante Produkte bauen, beispielsweise eine Karte mit Burgen und Schlösser, die mit Sachdaten aus Wikidata angereichert wird. Die Qualität dieser manuell erfassten Verknüpfungen ist bislang jedoch unbekannt und ungeprüft.

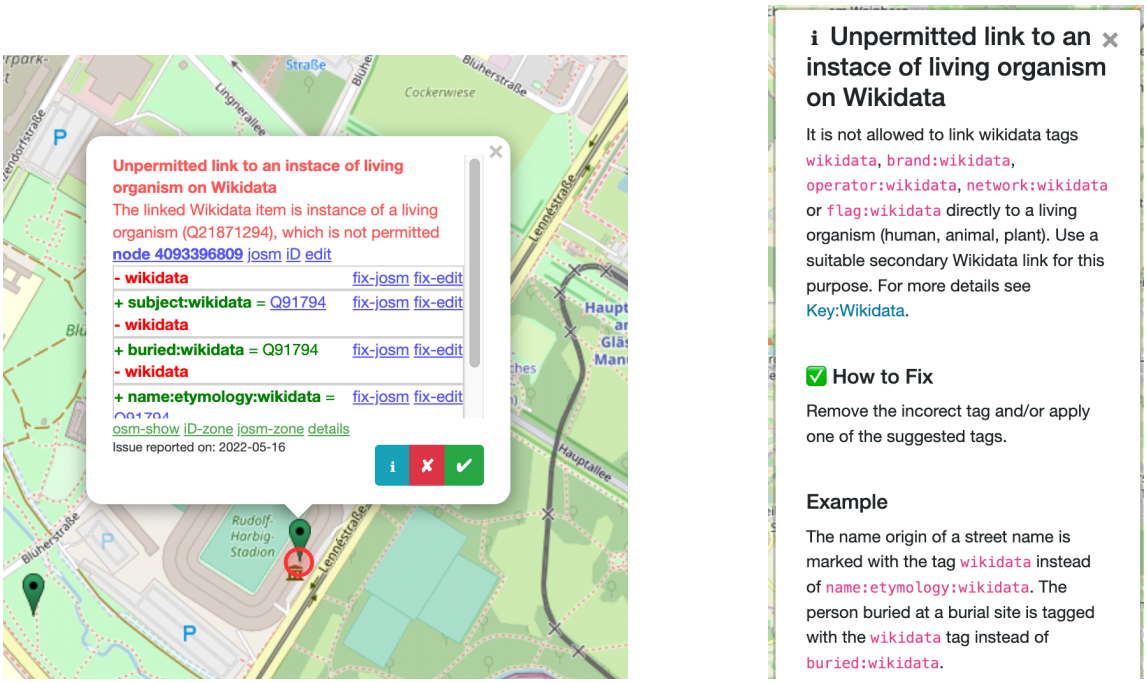


(a) Rapperswil auf OpenStreetMap mit Wikidata-Tag (b) Rapperswil in Wikidata

Abbildung 1: Beispiel einer Ortschaft mit Verknüpfung von OpenStreetMap nach Wikidata

Ziel der Arbeit

Es soll ein Werkzeug entwickelt werden, welches die bestehenden Verknüpfungen von OpenStreetMap nach Wikidata prüft. Die gefundenen Fehler werden mit einem Korrekturvorschlag an die externe Fehlerdatenbank Osmose gesendet. Die Fehler werden dort, wie in Abbildung 2 sichtbar, auf einer Karte dargestellt. Ziel ist es, dass die Applikation ein dauerhafter Teil der Infrastruktur zur Qualitätssicherung von OpenStreetMap wird. Sie muss mit den grossen Datenmengen der beiden Datenbanken (je ca. 1.5 TB) zurechtkommen und die wöchentlich erscheinenden Datenbank-Dumps innerhalb dieser Frist verarbeiten können. Ausserdem ist auf gutes Software-Engineering und die Code-Qualität zu achten, sodass das Tool gewartet und weiterentwickelt werden kann.



(a) Anzeige und Korrekturvorschläge (b) Zusatzinformationen

Abbildung 2: Anzeige eines Fehlers im Osmose-Frontend

Resultate

Die entwickelte Applikation `osm wikidata quality checker` läuft in einem Docker-Container, besorgt sich selbstständig die Datensätze, verarbeitet diese und sendet die gefundenen Fehler am Schluss an das Osmose-Frontend. Durch den Einsatz von Multiprocessing und des entwickelten Datenbankmodells, bei dem nur die relevanten Daten extrahiert werden, ist es in der Lage, die grossen Datenmengen zu verarbeiten. Die Laufzeit des Programms beträgt von Anfang bis Ende rund einen Tag auf dem Referenzsystem. Auch auf schwächeren Servern kann eine Laufzeit von unter einer Woche eingehalten werden. In Abbildung 3 ist der Ablauf und Datenfluss der entstandenen Applikation zu sehen.

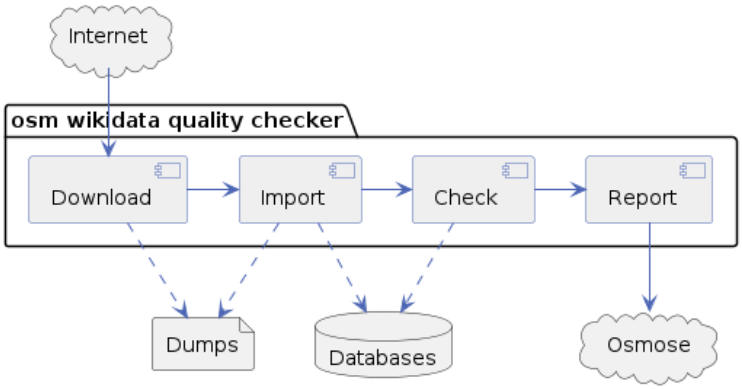


Abbildung 3: Ablauf und Datenfluss der Applikation `osm wikidata quality checker`

Auch Schwierigkeiten im Umgang mit crowdsourced Data, bei denen mit unvorhergesehene Datenfehlern gerechnet werden muss, wurden erfolgreich gemeistert, sodass eine hohe Fehlertoleranz erreicht wurde. Eine ausführliche Dokumentation sowie die leicht verständliche Architektur

ermöglicht es, das Tool auszubauen und weitere Checks zu implementieren. Die optionale Konfiguration bietet die nötige Flexibilität im Betrieb und hilft bei der Weiterentwicklung.

Aktuell werden weltweit über 30’000 Fehler in den in Abbildung 4 sichtbaren neun Kategorien gefunden. Dabei wird eine hohe Treffsicherheit von 95% erreicht.

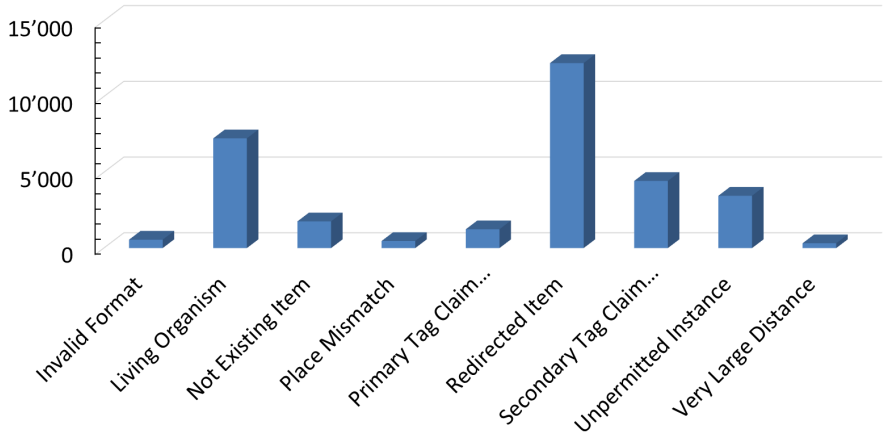


Abbildung 4: Gefundene Fehler nach Kategorie

Fazit und Ausblick

Die gesetzten Ziele wurden alle erreicht und die Applikation liefert einen wertvollen Beitrag zur Qualitätssicherung auf OpenStreetMap. Entstanden ist ein betriebsreifes Werkzeug, welches selbstständig läuft und die gefundenen Fehler an das Osmose-Frontend sendet. Es ist in der Lage, mit den grossen Datenmengen umzugehen und die gesamte Welt innerhalb der geforderten Frist zu prüfen.

Weiterhin gibt es viele denkbare Erweiterungen, um den Funktionsumfang und die Genauigkeit der Checks zu erhöhen.

Danke

Danke an alle Unterstützer dieser Arbeit:

Unser erstes Dankeschön geht an unseren Betreuer, Prof. Stefan Keller, für die spannende und herausfordernde Arbeit sowie seine wertvollen Beiträge und Ideen.

Ein spezieller Dank geht an unseren Projektpartner, Sascha Brawer, der uns über die gesamte Dauer der Arbeit begleitet und uns mit hilfreichen Ratschlägen und wertvollem Fachwissen im Bereich OpenStreetMap und Wikidata unterstützt hat.

Bei Janine Elmer bedanken wir uns ganz herzlich für das Korrekturlesen der gesamten Arbeit.

Zusätzlich bedanken wir uns bei Frédéric Rodrigo von der Osmose-Community, der uns im Bereich der Datenlieferung an das Osmose-Frontend unterstützt hat.

Zum Schluss geht unser Dank an unsere Familien, Partner und Freunde, für die positive Unterstützung und Geduld.

Inhaltsverzeichnis

Glossar

bzip2 ist ein freies Komprimierungsprogramm zur verlustfreien Kompression von Dateien. ix, 8, 12

CSV ist ein Dateiformat, bei dem einzelne Felder durch Kommas oder einen definierten Character separiert werden. ix, 12, 23

Docker Kann Anwendungen mit Hilfe von Containervirtualisierung isolieren. Die daraus entstehenden Container können einfach bereitgestellt werden, da alle Abhängigkeiten darin untergebracht sind. ix, 18, 40, 51, 74

False Negative Ein False Negative ist ein zu Unrecht nicht erkannter Fehler. ix, 17, 32, 33, 50

False Positive Ein False Positive ist ein zu Unrecht gemeldeter Fehler. ix, 12, 13, 17, 26, 31–35, 46, 50, 53, 59

GitLab GitLab ist eine Webanwendung zur Versionsverwaltung von Softwareprojekten. Es bietet unter anderen ein Issue-Tracking-System mit Kanban-Board und ein System für Continuous Integration (CI). ix, 6, 57, 58, 74

GPL (GNU General Public License) ist eine Softwarelizenz, die einem erlaubt, die betreffende Software auszuführen, zu studieren, zu verändern und zu verbreiten. ix, 12

iD Editor Einfacher plattformunabhängiger OpenStreetMap-Editor, der im Browser läuft. ix, 15

JOSM Der Java OpenStreetMap Editor ist eine Desktop-Applikation, um OpenStreetMap zu bearbeiten. ix, 15

JSON Die JavaScript Object Notation (JSON) ist ein kompaktes und lesbares Dateiformat, welches für den Austausch von Daten zwischen verschiedenen Systemen geeignet ist. ix, 12, 36

Osmose Osmose ist eines der vielen Qualitätssicherungswerkzeuge, die zur Erkennung von Problemen in OpenStreetMap-Daten zur Verfügung stehen. Es erkennt ein sehr breites Spektrum an Fehlerarten. i, ix, 6, 20, 23, 36, 46, 47, 54, 55, 62

Overpass-Turbo Webtool um Overpass-API-Queries auf den OSM-Daten auszuführen und deren Resultat anzeigen zu lassen. ix, 15

PBF Protocolbuffer Binary Format (Protobuf) ist ein freies und quelloffenes Datenformat, das zur Serialisierung strukturierter Daten verwendet wird. ix, 37, 47

PostgreSQL PostgreSQL ist ein freies, objektrelationales Datenbankmanagementsystem. Wird oft kurz Postgres genannt. ix, 8, 9

Primary Wikidata-Tag Ist ein Tag in OpenStreetMap, welches direkt auf das entsprechende Objekt in Wikidata zeigt. ix, 9, 82

Python ist eine universelle und interpretierte Programmiersprache. Sie läuft auf den gängigen Betriebssystemen.. ix, 6, 47

Secondary Wikidata-Tag Ist ein Tag in OpenStreetMap, welches Wikidata nur als Postfix enthält mit dem Objekt zusammenhängende Information trägt. ix, 28

SPARQL ist eine graphenbasierte Abfragesprache für Abfragen von Inhalten aus Datenbanken wie Wikidata. Sie wird zur Formulierung logischer Aussagen über beliebige Dinge genutzt. ix, 12, 15, 27, 84

SQLite ist eine Library, die ein relationales Datenbanksystem implementiert. Es ist das verbreitetste und meistverwendete Datenbanksystem. ix, 47

True Positive Ein False Positive ist ein zu Recht gemeldeter Fehler. ix, 17, 28, 29, 32–34

Well-Known Binary (WKB) Format, um geographische und geometrische Daten zu repräsentieren. ix, 36, 39, 47

Wikidata Query Service stellt ein SPARQL-Endpoint mit einer leistungsstarken Benutzeroberfläche zur Verfügung. ix

XML XML (Extensible Markup Language) ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten als Textdatei. Das Format ist sowohl von Menschen als auch von Maschinen lesbar. ix, 13, 46, 50

Akronyme

API Application Programming Interface. ix, 24, 47, 54

CI Continuous Integration. ix, 48, 53, 57–59, 74
CLI Command Line Interface. ix, 40

GUI Graphical User Interface. ix, 17, 18

IFS Institut für Software. ix, 3, 6

ORM Object Relational Mapping. ix, 47
OSM OpenStreetMap. ix, 9, 11, 12, 14–16, 20–30, 32–39, 43–47, 53, 61, 62, 82
OST Ostschweizer Fachhochschule. ix, 3, 6, 11, 17

PEP Python Enhancement Proposals. ix, 59
POI Point of Interest. ix, 6, 54

SQL Structured Query Language. ix, 22, 47

WD Wikidata. ix, 9, 20–23, 43, 44, 61, 62

KAPITEL 1

Einleitung

In diesem Kapitel werden die Ausgangslage, das Ziel der Arbeit und deren Rahmenbedingungen aufgezeigt. Der letzte Abschnitt gibt einen Überblick über die Struktur der folgenden Kapitel.

1.1 Ausgangslage

Die Ausgangslage beschreibt die angetroffene Problemstellung und der aktuelle Stand der Technik. Zudem gibt es eine Einführung in die Themenfelder der offenen und strukturierten Daten.

1.1.1 Problemstellung

OpenStreetMap ist eine freie Landkarte der ganzen Welt mit rund 1 Milliarde geografischen Objekten, wovon rund 20% mit Tags (Schlüssel-Wert-Paare) versehen sind. Diese tragen Informationen über das Objekt, wie zum Beispiel den Namen einer Ortschaft oder die Höchstgeschwindigkeit einer Strasse.

Wikidata ist eine freie Wissensdatenbank mit strukturierten Information über 97 Millionen[1] real oder fiktiv existierender Dinge.

Während OpenStreetMap vorwiegend geografisch relevante und aktuell gültige Informationen enthält, sind in Wikidata auch multimediale und historische Information vorhanden, wie zum Beispiel das Wappen einer Stadt oder eine Liste aller Bürgermeister mit deren Amtsperioden.

Über Wikidata-Tags können Objekte auf Wikidata Einträge verweisen. Mittels dieser Verknüpfungen lassen sich interessante Produkte bauen, beispielsweise eine Karte mit Schweizer Städten, welche mit Sachdaten aus Wikidata angereichert wird. Allerdings sind die meisten Wikidata-Tags in OpenStreetMap von Hand gesetzt und die Qualität der Daten unbekannt.

In dieser Arbeit soll ein Werkzeug, der *OpenStreetMap Wikidata Quality Checker*, entwickelt werden, welches die Verknüpfungen von OpenStreetMap zu Wikidata überprüft und somit die Datenqualität von OpenStreetMap verbessert und sichert.

1.1.2 Crowdsourced Open Data

Frei nutzbare und öffentlich zugängliche Informationen werden als Open-Data bezeichnet. Ein Spezialfall stellen dabei Crowdsourced-Open-Data, auch Citizen-Science genannt, dar (See et al., 2016), wo diese Daten nicht nur frei verfügbar sind, sondern auch von jedermann selbst erstellt und geändert werden können. Dieses Recht auf universelle Beteiligung stellt ein wichtiger Grundsatz dar:

> *Jeder muss in der Lage sein, die Daten zu nutzen, zu verarbeiten und weiterzuverteilen - es darf keine Benachteiligung von einzelnen Personen, Gruppen, oder Anwendungszwecken geben.(Open Knowledge Foundation, 2022)*

Da diese Daten von grossen Gruppen Freiwilliger zusammengetragen und gepflegt werden, gibt es keine Garantie auf Vollständigkeit oder Richtigkeit. Verschiedene Menschen haben unterschiedliche Vorstellungen und Standards. Dies widerspiegelt sich auch bei Crowdsourced-Open-Data und kann Auswirkungen auf die Datenqualität haben. Es ist bei solchen Projekten deshalb zwingend, die von der Community im Konsens einwickelten Richtlinien zu überprüfen.

1.1.3 Strukturierte Daten

Strukturierte Daten sind Informationen, welche in einem definierten Schema abgelegt sind. Somit lassen sich diese, im Gegensatz zu unstrukturierten Daten, leicht automatisiert und maschinell verarbeiten. Ein weiterer Vorteil ist, dass explizite Verknüpfungen von Informationen untereinander möglich sind. So ist ein Traversieren der Informationen und das Erstellen von Auswertungen,

Listen oder Baumstrukturen möglich. Anhand der Tabelle 1.1 soll der Unterschied zwischen strukturierten und unstrukturierten Daten erläutert werden.

Beispiel	**Einordnung**	**Datenformat und Verarbeitung**	**Maschinelle Verarbeitung**
Buch	unstrukturiert	Freitext. Es muss der gesamte Text durchgegangen und interpretiert werden.	Sehr schwer
Lexikon	semi-strukturiert	Alphabetisch sortiert und mit Index, jedoch Freitext bei den jeweiligen Begriffen.	schwer
Quartettkarten	strukturiert	Die Werte zu den Spielkarten sind alle mit derselben Struktur abgelegt.	leicht

Tabelle 1.1: Beispiele strukturierter und unstrukturierter Formate

1.1.4 Stand der Technik

Crowdsourced Open-Data-Projekte wie OpenStreetMap und Wikidata werden immer populärer. Gleichzeitig wird eine stärkere Verknüpfung solcher Projekte angestrebt, so wie das beim Wikidata-Tag auf OpenStreetMap der Fall ist.

Wie es der Name ”offene Daten” besagt, kann grundsätzlich jeder diese Daten verändern, löschen oder ergänzen. Um die Qualität sicherzustellen, gibt es verschiedene automatische Analyse-Tools, welche die Veränderungen auf Fehler überprüfen. Ein Überblick der wichtigsten existierenden Tools ist in Unterabschnitt 2.1.5 zu finden. Aktuell existiert jedoch kein Analyse-Tool, welches die Verknüpfungen von OpenStreetMap und Wikidata überprüft. Die Datenqualität der Verknüpfung zwischen OpenStreetMap und Wikidata ist somit unbekannt.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist, bestehende Verknüpfungen von OpenStreetMap zu Wikidata zu verifizieren, um so die Datenqualität auf OpenStreetMap zu erhöhen. Dazu soll ein Tool entwickelt werden, welches automatisiert die vorhandenen Verknüpfungen analysiert und die gefunden Fehler meldet. Die detaillierte Aufgabenstellung befindet sich im Anhang A.

1.3 Rahmenbedingungen

Die folgende Arbeit wurde im Rahmen einer Bachelorarbeit an der Ostschweizer Fachhochschule (OST) im Auftrag des Institut für Software (IFS) durchgeführt. Die Bachelorarbeit wird jedem Studenten mit 12 ECTS vergütet, was einen Zeitrahmen von je 360 Stunden entspricht. Die Arbeit fand im Zeitraum vom 21. Februar bis 17. Juni 2022 statt.

1.4 Aufbau der Arbeit

Die nachfolgende Zusammenfassung gibt einen Überblick über den Zweck und Inhalt der kommenden Kapitel:

Kapitel 2 Analyse beinhaltet die Untersuchung der Problemstellung und des Systemkontextes, welcher aus OpenStreetMap, Wikidata und der Fehlerdatenbank Osmose besteht. Daraus ergeben sich die funktionalen und nicht-funktionalen Anforderungen an die Applikation.

Kapitel 3 Design beschreibt die Architektur und den Ablauf des Programm. Relevante Bereiche wie die Checks, das Datenbankmodell, der Umgang mit grossen Daten und das Deployment werden im Detail betrachtet.

Kapitel 4 Implementation erklärt relevante Aspekte bei der Umsetzung des Designs wie Checks, Import und Osmose Reporter. Zum Schluss folgen die externen Libraries und das Testing.

Kapitel 5 Resultate fasst zusammen, was in dieser Arbeit erreicht wurde. Darauf basierend wird ein Ausblick gegeben. Dieser beschreibt, welche Probleme noch ungelöst sind und wie das Programm erweitert werden könnte.

Kapitel 6 Schlussfolgerung reflektiert das Erreichte und gibt eine Interpretation der Resultate.

Kapitel 7 Projekt Management schildert das Vorgehen, die Verantwortlichkeiten und den Ablauf des Projektes. Der Zeitplan ist durch ein Gantt-Projektplan dargestellt und die Meilensteine werden genauer beschrieben. Zudem sind die Risikoanalysen und Zeitauswertungen hier untergebracht.

KAPITEL 2

Analyse

In diesem Kapitel wird betrachtet, in welchem Kontext sich die Applikation befindet. Die Datenquellen und Umsysteme werden analysiert und in Kombination mit der Aufgabenstellung die funktionalen und nicht-funktionalen Anforderungen abgeleitet.

2.1 Systemkontext

Der Systemkontext beschreibt die Umgebung, in der sich das System befindet und wovon es beeinflusst wird. Zuerst werden die vorgegebenen Technologien betrachtet. Danach folgt je eine Analyse zu OpenStreetMap, Wikidata und des Qualitätssicherungs-Tools Osmose. Eine Analyse des Marktumfeldes zeigt, welche vergleichbaren Anwendungen existieren.

2.1.1 Technologische Vorgaben

Ausgehend von der Aufgabenstellung (Anhang A) sind die folgenden Technologien festgelegt.

Programmiersprache

Es ist vorgesehen, auf die Programmiersprache Python zu setzen. Python eignet sich hervorragend für solche Projekte, da es bereits etliche Libraries (siehe Unterabschnitt 4.4.1) für die Verarbeitung der gegebenen Rohdaten gibt und ein schnelles Prototyping möglich ist. Es ist jedoch auch erlaubt, Go oder C++ für das gesamte Projekt oder Teile davon einzusetzen.

Infrastruktur

Für die gesamte Entwicklungszeit steht die GitLab-Instanz der OST[2] zur Verfügung. Der finale Stand ist auf einem öffentlichen GitLab-Repository[3] publiziert.

Für Entwicklung und Betrieb der Applikation steht ein Server des IFS zur Verfügung, welcher die nötige Kapazität besitzt, um die grossen Datenmengen zu verarbeiten.

Lizenz

Das Tool wird unter einer open-source MIT-Lizenz veröffentlicht. Verbesserungen und Vorschläge für neue Funktionen sind erwünscht.

2.1.2 OpenStreetMap

OpenStreetMap[4] ist ein Open-Source Projekt, welches frei nutzbare Geoinformationen sammelt und für jedermann kostenlos zur Verfügung stellt. Kern des Projekts ist eine offen zugängliche Datenbank mit den gesammelten Geodaten. Die Rohdatengewinnung erfolgt mit Hilfe einer grossen weltweiten Community, welche mit ihrem lokalen Wissen laufend Daten erweitert, ergänzt oder korrigiert. Aus diesen Daten können mit sogenannten Overlays diverse Landkarten generiert werden, wie Strassenkarten oder eine Karte mit Radwegen. Aber auch Karten für humanitäre Projekte oder mit Point of Interests (POIs) für Notsituationen wie Wasserressourcen, öffentlichen Gebäuden oder medizinischen Einrichtungen oder lassen sich damit erstellen.

Durch eigene Overlays und Verknüpfung mit externen Datenquellen lassen sich weitere Interessante Projekte bauen und in Webseiten oder Applikationen einbinden. Ein Beispiel dafür ist das Projekt Burgen-Dossier-Karte Schweiz[5].

Datenstruktur und Modellierung

In OpenStreetMap werden Dinge mithilfe den in Tabelle 2.1 beschriebenen Objektarten modelliert. Durch diese drei Spezialisierungen ist es möglich, sämtliche Objekte der Welt abzubilden. Die Relations folgen dabei einem Art Composite-Pattern mit beliebiger Tiefe. Zurzeit gibt es etwa 6.9 Milliarden Nodes, 768 Millionen Ways und 8.9 Millionen Relationen. (Wikipedia, 2022b)

Objekt	Definition	Beispiele
Node	Ein Punkt auf der Karte mit Koordinate (Lat/Lon)	Restaurant, Grenzstein
Way	Eine Sammlung von Node's, welche eine offene oder geschlossene Linie (Polygon) bilden	Strasse, Gebäudeumriss, Fluss, Grenzabschnitt
Relation	Eine Sammlung von Node's, Way's oder weiteren Sub-Relationen	Autobahnnetz, Zugstrecke, Campus, Ortschaft

Tabelle 2.1: Objektarten in OpenStreetMap

Die Abbildung 2.1 zeigt, wie die Stadt Rapperswil als Relation modelliert ist. Diese besteht aus einem Knoten als repräsentativer Punkt und 11 Way's, welche die Stadtgrenze bilden. Da die Grenze von Rapperswil auch die Kantonsgrenze bildet, sind Teile davon auch in der Relation des *Kantons St. Gallen* (Relation 1687006) enthalten.

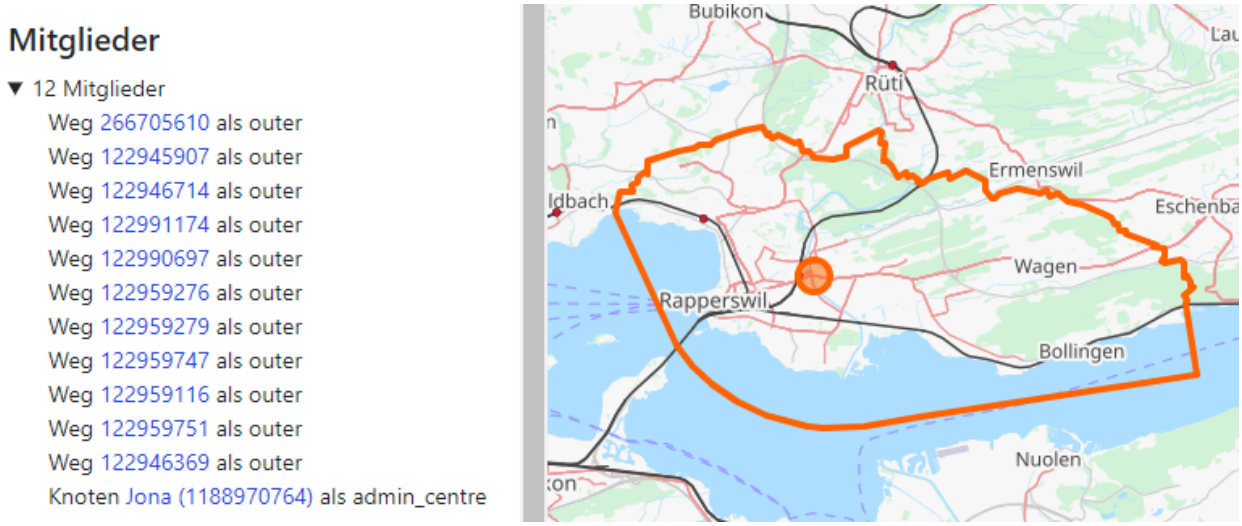


Abbildung 2.1: Mitglieder der Relation "Rapperswil"

Tag's

Jedes der drei Objektarten (Node, Way, Relation) kann mit Merkmalen, sogenannten Tag's, angereichert werden. Ein Tag ist ein frei wählbares Schlüssel-Wert-Paar und wird als `key=value` dargestellt. Dabei können die Schlüssel mit Präfixen, Infixen oder Suffixen ergänzt werden, um diese genauer zu spezifizieren. Zum Beispiel kann mit einem Sprach-Suffix ein Ortsname ihn verschiedenen Sprachen angegeben werden (`name:de=*`). Mehre Werte können mit Semikolon getrennt werden (`key=value1;value2`), dies ist jedoch selten der Fall. (OpenStreetMap, 2021b).

Welches Tag für was verwendet wird, ist von der Community bestimmt. (OpenStreetMap, 2021a) Für eine Ortschaft sind zum Beispiel Eigenschaften wie der Name und die Einwohnerzahl interessant. (OpenStreetMap, 2021e). In der Abbildung 2.2 ist zu sehen, welche Tags der Knoten Rapperswil besitzt.

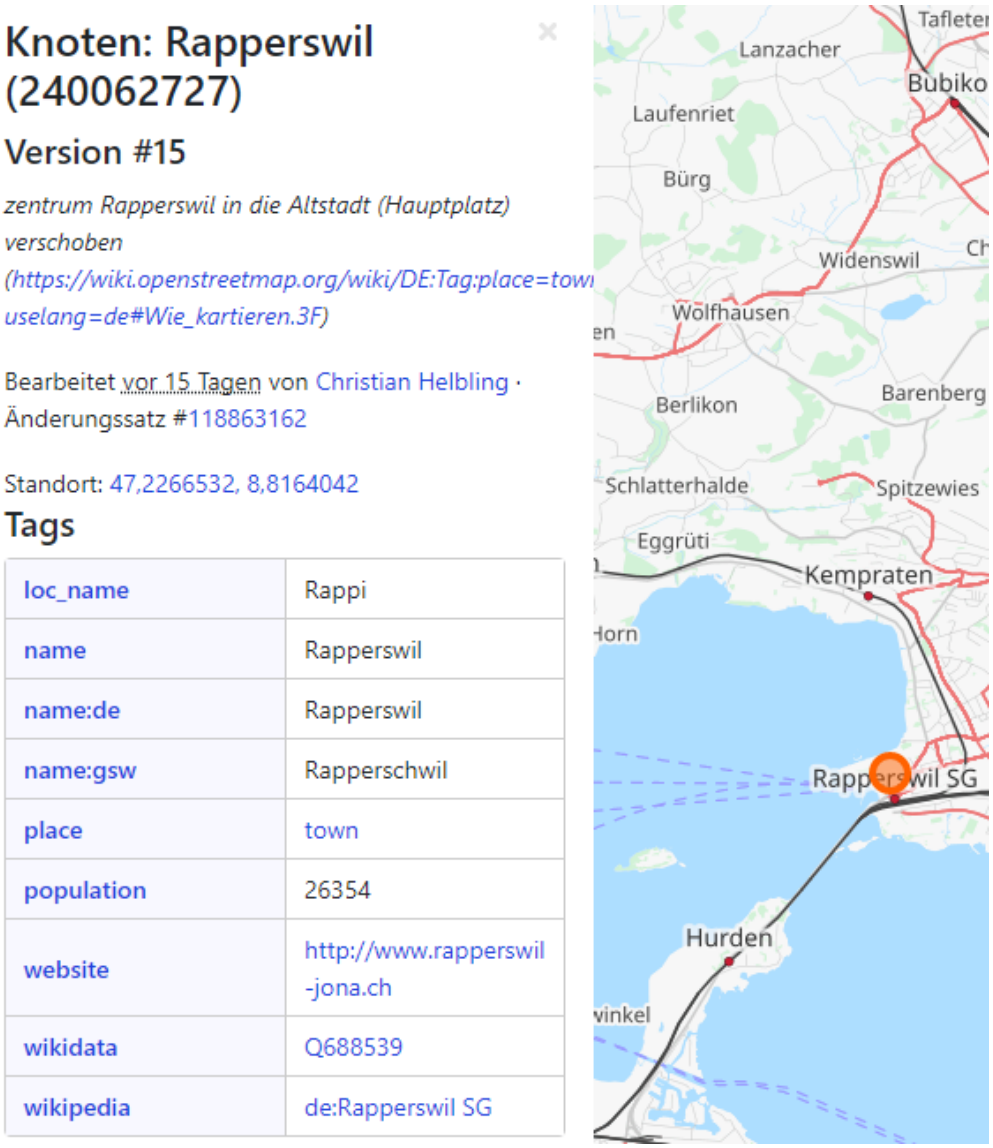


Abbildung 2.2: Knoten ”Rapperswil” (Ortsmittelpunkt) mit Eigenschaften (Tag’s)

Diverse Quality Assurance Tools kontrollieren die Einhaltung der von der Community vorgegebenen Richtlinien. So soll zum Beispiel ein Objekt, welche als Eigenschaft `shape=polygon` angegeben hat auch aus geschlossenen Wegen bestehen. Es herrscht jedoch in der Community nicht immer Einigkeit und es gibt verschiedene Möglichkeiten, ein Objekt zu modulieren. Ausserdem können sich Richtlinien mit der Zeit ändern, wie das folgende Zitat zeigt:

> *Historically, especially in Germany, `type=multipolygon` has been used instead of `type=boundary` for boundary relations. This method was not widely accepted and should be considered deprecated. (OpenStreetMap, 2022f)*

Datenformat und Download

Die OpenStreetMap Datenbank wird auf PostgreSQL mit der PostGIS Erweiterung für geospezifische Funktionen betrieben (OpenStreetMap, 2021d). Die in der Datenbank enthaltenen Informationen werden in einem eigenen, als *OSM XML* bezeichneten Format[6]zur Verfügung gestellt. Diverse Mirrors bieten die gesamte Welt im Originalformat oder als *bzip2* komprimiert zum Download an. Weitere Dienste bieten regionale oder länderspezifische Dumps an (OpenStreetMap, 2022e).

Nach dem Download können die Dumps in eine eigene PostgreSQL-Instanz importiert werden, oder mittels anderen Tools wie zum Beispiel Pyosmium (Unterabschnitt 4.4.1) verarbeitet werden.

Verknüpfung mit Wikimedia

Zwischen OpenStreetMap und Wikimedia findet seit längerem eine Zusammenarbeit statt. Von beiden Seiten aus können Verknüpfungen modelliert werden, sodass sich die beiden freien Projekte optimal ergänzen.

Für die Verbindung von OpenStreetMap (OSM) zu Wikidata (WD) steht neben den weit verbreiteten Tag `wikipedia` für Wikipedia-Artikel, seit 2014[7] auch das Tag `wikidata` zur Verfügung. Gut zu sehen ist in der Abbildung 2.3, dass die Anzahl der Verknüpfungen im 2016 sprunghaft angestiegen ist und seitdem nicht an Popularität verloren hat.

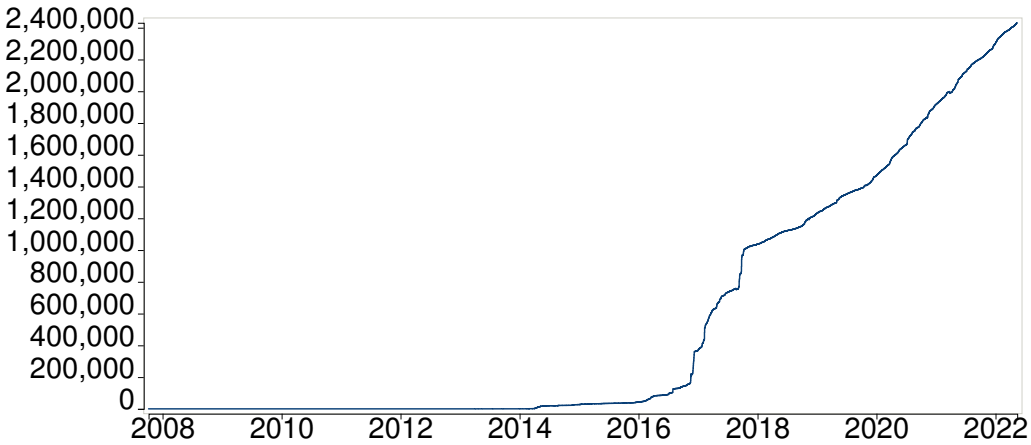


Abbildung 2.3: Chronology des Primary Wikidata-Tag auf OSM
Quelle: `https://taginfo.openstreetmap.org/keys/wikidata#chronology`

Wie die anderen Tags, können auch Wikidata-Tags mit Präfixen versehen werden. So kann zum Beispiel mit `brand:wikidata=*` angegeben werden, zu welcher Handelskette ein Geschäft gehört. Momentan gibt es über 5 Millionen Objekte mit einem Wikipedia-Tag. Davon sind rund die Hälfte mit Präfixen versehen, nachfolgend Secondary Tags genannt. Die Tabelle 2.2 zeigt die am häufigsten vorkommenden Wikidata-Tags.

Tag	**Anzahl**
wikidata	2’440’344
brand:wikidata	1’472’636
operator:wikidata	763’840
network:wikidata	356’661
name:etymology:wikidata	323’380
species:wikidata	26’077
subject:wikidata	25’501
flag:wikidata	12’239
genus:wikidata	7’659
artist:wikidata	4’789
architect:wikidata	4’412
Total	**5’467’708**

Tabelle 2.2: Die wichtigsten Wikidata-Tags
Quelle: `https://taginfo.openstreetmap.org/keys/wikidata`

2.1.3 Wikidata

Wikidata[8] ist eine große und freie Wissensdatenbank, die von jedermann bearbeitet werden kann. Durch die Bereitstellung von strukturierten Daten (siehe Unterabschnitt 1.1.3), kann Wikidata unter anderem andere Projekte unterstützten in dem zum Beispiel Geburtsdaten von Personen, Listen von Ländern oder sonstige allgemeingültige Daten in allen Artikeln der Wikimedia-Projekte verfügbar macht.

Seit dem Start 2012 verzeichnet Wikidata ein starkes Wachstum. Gemäss den Statistiken von Wikidata (2022b) sind mittlerweile über 97 Millionen Datenobjekte vorhanden. Die Daten in Wikidata stehen unter der Creative-Commons-Lizenz CC0 1.0 Universal (CC0 1.0) Public Domain Dedication.

Datenobjekte

Informationen werden in Wikidata in Datenobjekten, den Items, gesammelt. Jedes Item besitzt eine eindeutige Kennung, bestehend aus dem Präfix ”Q” gefolgt von einer Zahl, wie zum Beispiel Q123. Grundsätzlich kann alles in der Welt Vorkommende ein Item sein. Dabei muss es nicht physisch existieren, auch abstrakte Objekte wie Zitate, Familiennamen, Wörter und Ideen können erfasst werden. Dasselbe gilt für Metadaten wie Kategorien oder Taxonomien. Auch Listen von Dingen können erstellt werden.

Um Items treffend zu bezeichnen und zu unterscheiden, können ihnen Namen, eine kurze Beschreibung sowie verschiedene Aliasse hinzugefügt werden. Da Wikidata mehrsprachig ist, können alle diese Texte in mehreren Sprachen erfasst werden. Die Tabelle 2.3 zeigt Wikidata-Items, welche auf Grund ihres ähnlichen Namens nur durch die Q-Id oder der Beschreibung auseinander gehalten werden können.

Q-Id	Bezeichnung	Beschreibung
Q1248784	Airport	Location where aircraft take off and land
Q15114	Zurich Airport	International airport serving Zurich
Q409022	Airport	1970 American disaster-drama film ...
Q697292	AirPort	Brand of wifi router equipment made by Apple Inc.

Tabelle 2.3: Beispiel verschiedener Wikidata-Objekte

Labels und Claims

Informationen über das Objekt werden mit Aussagen, welche auch Claims genannt werden, erfasst. Im Grunde sind dies Schlüssel-Wert-Paare, bestehend aus einer Eigenschaft (*Propertyname* (P123)) und einem oder mehreren dazugehörenden Werten. Je nach Property sind unterschiedliche Wertetypen erlaubt, wie Zahlen, Freitext, Links, Bilder, Multimedia-Objekte oder eine Verknüpfung auf ein anderes Wikidata Item. Die Tabelle 2.4 zeigt Beispiele solcher Claims und deren Datentypen.

Property	Datentyp	Beispiel für Wert
instance of (P31)	Wikidata-Item	*educational institution* (Q2385804)
image (P18)	Commons media file	Ost-campus_rapperswil_gebeaude1_eingang.jpg
inception (P571)	Point in time	2019
street address (P6375)	Monolingual text	Oberseestrasse 10, 8640 Rapperswil SG

Tabelle 2.4: Beispiele von Aussagen (Claims) über die OST

Die bei fast jedem Item vorhandenen Properties *instance of* (P31) und *subclass of* (P279) sind hierarchisch aufgebaut und ergeben eine Baumstruktur. Den Claims können optional Quellen oder Links zu weiteren Wikimedia-Projekten hinzugefügt werden. Abbildung 2.4 zeigt wie die OST auf Wikidata als Objekt erfasst. Zu sehen sind die Labels sowie die ersten drei Claims.

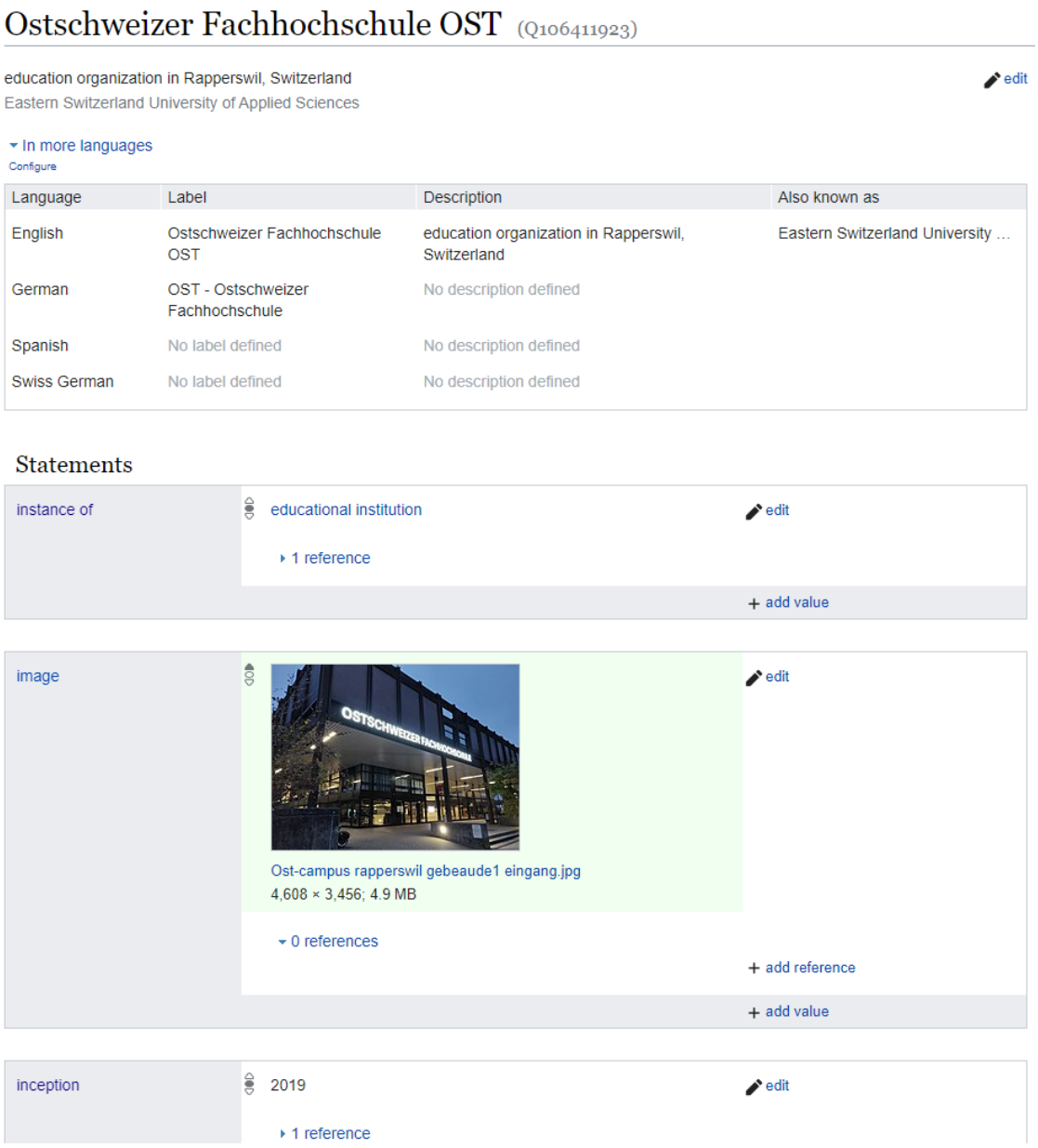


Abbildung 2.4: Ausschnitt der Wikidata-Seite der OST

Relevante Properties

Eine Reihe von Properties sind für diese Arbeit relevant, da sie für die Überprüfung der Verknüpfung verwendet werden können.

Ein sehr wertvolles Property stellt *coordinate location* (P625) dar, mit welchem einem Koordinaten hinzugefügt werden können und so indirekt eine Verbindung zu diversen Kartendiensten und somit auch OpenStreetMap entsteht.

Diverse weitere Properties wie der Name (Label), *street address* (P6375) oder auch die Kategorisierungen über *instance of* (P31) können genutzt werden, um Objekte in OSM zu Identifizieren.

Tools wie der *OpenStreetMap Wikidata Matcher* (siehe Abschnitt 2.1.5) nutzen diese Möglichkeit.

Mit dem Property *OpenStreetMap tag or key* (P1282) kann angegeben werden, welche OSM-Tags ein Wikidata-Objekt hat. Zum Beispiel sollte ein verknüpftes Wikidata-Objekt der Klasse *Werft* (Q190928) auf OSM das Tag `waterway=boatyard` oder `industrial=shipyard` haben.

Das Property *OpenStreetMap relation ID* (P402) war als direkte Verknüpfung zum OSM-Objekt gedacht. Von dessen Verwendung wird jedoch abgeraten, da die OSM-Id's nicht stabil sind und sich theoretisch ändern können. (Wikidata, 2022a)

Datenzugriff

Wikidata stellt ein Query Service[9] zur Verfügung. Über diesen SPARQL Endpunkt können Abfragen auf die Wissensdatenbank gemacht werden.

Die komplette Datenbank wird als Dump im JSON Format zum Download angeboten. Der mit bzip2 komprimierte Dump ist zurzeit etwa 73 GB gross. Dekomprimiert ergibt das einen Datensatz von rund 1.5 TB.

Des Weiteren gibt es Libraries wie den *Wikidata Client Library* (siehe Unterabschnitt 4.4.3), die den Datenzugriff aus Python heraus ermöglicht.

Redirects

Es kommt vor, dass für dasselbe Ding versehentlich zwei oder mehrere Items erstellt werden. Mit Redirects werden diese wieder zusammengeführt und die Q-Id der doppelten Objekte wird zur Hauptinstanz umgeleitet. Zurzeit gibt es etwa 3.7 Millionen Redirects. Diese Redirects sind in der Wikipedia-Datenbank gespeichert, jedoch nicht in den Dumps enthalten (Hoch, 2014).

Um eine Liste von Redirects zu erhalten, kann ein entsprechende SPARQL-Query ausgeführt werden. Die Abfrage der gesamten Liste muss paginiert werden und nimmt einiges an Zeit in Anspruch. Der Projektpartner Sascha Brawer hat einen Service geschrieben, welcher die aggregierte Liste als CSV-File unter dem Link `https://qrank.wmcloud.org/download/qredirects.csv.gz` zur Verfügung stellt. (Brawer, User:Infrastruktur & User:Dipsacus fullonum, 2022)

Durch die einzeln ausgeführten Abfragen auf der Live-Datenbank, kann es bei gleichzeitigen Mutationen gelegentlich vorkommen, dass ein Redirect nicht oder mehrfach in der Liste vorkommen. Dieser Umstand muss beim Verarbeiten der Liste beachtet werden.

2.1.4 Osmose

Gemäss Basiri, Haklay, Foody und Mooney (2019) ist es bei Corwdsourced-Geographical-Data wichtig, ein Verständnis für die vorliegende Qualität zu erhalten. Osmose[10] ist ein solches Werkzeug zur Qualitätssicherung von OpenStreetMap. Es dient dazu, Fehler und Inkonsistenzen in den Daten zu erkennen. Diese werden von Hand von der Community geprüft und entweder behoben oder als False Positive zurückgewiesen. Der Name Osmose steht für "OpenStreetMap Oversight Search Engine". Der Quellcode steht unter GPL und ist auf GitHub frei verfügbar. (OpenStreetMap, 2022d)

Frontend und Analysers

Osmose besteht aus zwei Teilen. Einem Frontend, welches eine Datenbank mit den Fehlern führt und diese über ein Web-Interface den Benutzer darstellt, sowie mehreren Backends, sogenannten

Analysers, welche die OSM-Daten nach Fehler analysieren und diese ans Frontend melden. Das Frontend beinhaltet zudem einige Funktionen, die es dem Nutzer erleichtern, nach Fehler zu filtern und diese zu beheben.

XML-Format, Fehlerkategorien und Korrekturvorschläge

Die einzelnen Analysers des Backends generieren Listen von potentiellen Fehlern und melden diese in speziellen XML-Format[11], welches in Listing 2.1 zu sehen ist, an das Frontend.

Mögliche Fehler werden einer Item-Gruppe zugeordnet, welche im Frontend angewählt werden kann. Weiter können dem Issue noch Tags zugeordnet werden, auch diese dienen der Sortierung und Filterung im Frontend. Die letzte Hierarchiestufe ist die spezifische Klasse[12] mit eindeutiger Id und Titel, welche die individuellen Fehler eines Analysers einordnet.

```
<?xml version="1.0" encoding="UTF-8"?>
<analysers timestamp="2014-10-17T09:52:58Z">
  <analyser timestamp="2014-10-17T09:52:58Z">
    <class item="3210" tag="highway,tag,fix:chair" id="1" level="2">
      <classtext lang="en" title="noexit on way with multiple exits" />
    </class>
    <error class="1" subclass="1">
      <location lat="43.2899342" lon="5.3618622" />
      <node version="3" user="FrançoiseR" lat="43.2899342" lon="5.3618622"
      ↪  id="33806225">
        <tag k="noexit" v="yes" />
      </node>
      <fixes>
        <fix>
          <node id="33806225">
            <tag action="delete" k="noexit" />
          </node>
        </fix>
      </fixes>
    </error>
  </analyser>
</analysers>
```

Listing 2.1: Minimales Beispiel eines Osmose XML-Reports

Die einzelnen Fehler können frei auf der Karte platziert und optional an ein OSM-Objekt zugeordnet werden. Es können zudem weitere Informationen zum Fehler angegeben werden, diese sollten es der Community erleichtern, den Fehler zu beheben (Untertitel, Anleitung, Stolperfallen). Darunter fallen auch sogenannte Fixes, welche ja nach Kategorie *DELETE CHANGE* oder *NEW* ein Vorschlag zur Korrektur geben. Über das Level wird die Relevanz des Fehlers in drei Stufen (1: high, 2: normal, 3: low.) angegeben.

Die Abbildung 2.5 zeigt, wie Fehler im Frontend dargestellt werden. Diese können von der Community als behoben oder als *False Positive* zurückgewiesen werden

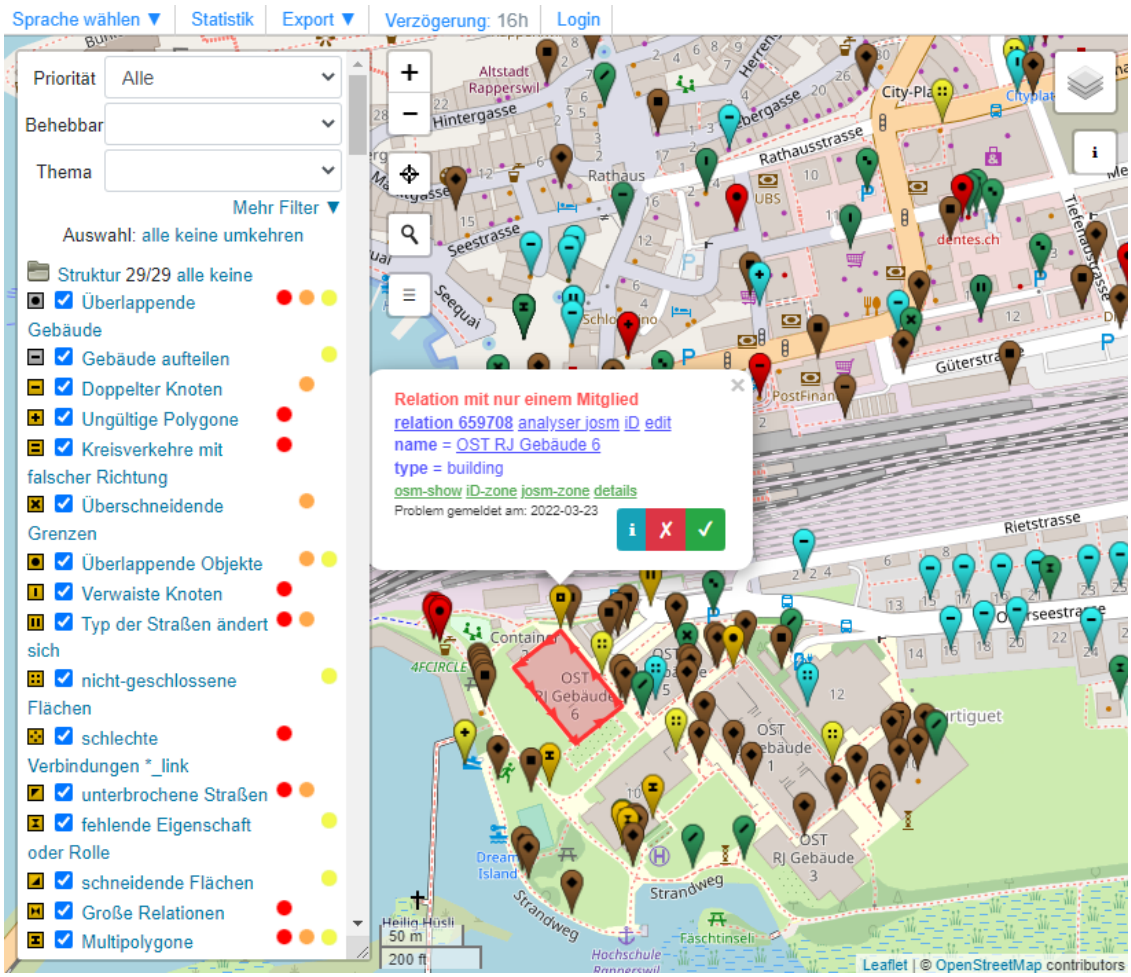


Abbildung 2.5: Screenshot von Osmose mit Detailansicht eines Fehlers

Lieferung von Fehler-Rapports

Während die im Backend integrierten Analysers ihre Fehler automatisch über das Framework an das Frontend melden, gibt es jedoch auch die Möglichkeit, Fehlerlisten eines externen Analysers direkt per HTTP POST an das Frontend zu melden. Dazu muss der Analysers im Frontend erfasst werden und seine individuelle Kennung (”source” genannt) sowie ein Passwort (”code” genannt) zur Identifizierung senden.

2.1.5 Vergleichbare Tools

In der Welt von OSM gibt es diverse Tools, welche für die Datenanreicherung oder Datenqualität zuständig sind. Einige dieser Tools drehen sich auch um Wikidata und Wikipedia, erfüllen jedoch nicht denselben Zweck wie der `osm wikidata quality checker` oder sind nicht mehr aktiv. Im Folgenden wird eine Übersicht dieser Tools gezeigt.

OSM - Wikidata matcher

Der OSM - Wikidata matcher[13] ist ein Webtool, welche dazu gedacht ist, OpenStreetMap-Objekte mit Wikidata-Tags anzureichern und so die Anzahl Verknüpfungen zu erhöhen. In einem semi-automatischen Prozess werden mögliche Übereinstimmungen gefunden, die nach einer manu-

ellen Überprüfung unkompliziert zu OSM hinzugefügt werden können. Es erkennt eine mögliche Übereinstimmung, in dem es diverse Kriterien wie Ort, Name und Eigenschaften in den OSM-Tags und Wikidata-Claims vergleicht. (OpenStreetMap, 2021c)

Das Tool dient dazu OSM mit Wikidata-Tags zu ergänzen in dem es sinnvolle Vorschläge macht. Es kann die Qualität der manuell erstellten Verknüpfungen jedoch nicht überprüfen.

Sophox

Sophox[14] ist eine Sammlung von Datendiensten, die mit OSM-Daten arbeiten und externe Datenquellen wie Wikidata nutzen. Der Dienst verwendet SPARQL als Abfragesprache. Sophox kann auch für die direkte OSM-Bearbeitung verwendet werden, oder als Herausforderung wie MapRoulette. Ähnlich wie Overpass-Turbo kann dieser Dienst direkt von JOSM aus genutzt werden.

Sophox enthält aufgrund von Platz- und Leistungsbeschränkungen keine Kopie der Wikidata-Daten mehr. Momentan fehlen in Sophox auch die meisten OSM-Daten, dies ist vermutlich nicht beabsichtigt (OpenStreetMap, 2022g). Das Tool scheint nicht mehr aktiv oder zumindest nicht mehr aktiv gewartet zu sein, der Nutzen ist somit sehr beschränkt.

WIWOSM (Wikipedia where in OSM)

WIWOSM[15] ist ein Tool, um für einen Wikipedia-Artikel geometrische Objekte aus OpenStreetMap anzuzeigen. Diese Objekte werden über einen passenden `wikipedia=*` Tag identifiziert. WIWOSM kann somit Wikipedia-Inhalte mit den Möglichkeiten eines geografischen Informationssystems verbinden.

Es werden jedoch nur Wikipedia-Tags und keine Wikidata-Tags unterstützt (OpenStreetMap, 2020b).

iD Editor (Wikipedia Link Improvement Project)

Auch der normale iD Editor beachtet schon gewisse Sachen im Zusammenhang mit dem Wikidata-Tag. So fügt der Editor im Rahmen des ”Wikipedia Link Improvement Project”[16] automatisch ein Wikidata-Tag hinzu, wenn ein Benutzer ein Wikipedia-Feld hinzufügt.

Das Projekt kümmert sich aber hauptsächlich um Wikipedia-Tags. Es werden keine Validierungen oder Qualitätschecks gemacht, sondern die Wikidata-Tags ungeprüft übernommen. (OpenStreetMap, 2020a).

OSMgadget

OSMgadget[17] ist ein Browser-Erweiterung, welche zum aktuellen Wikipedia-Artikel eine Box mit Infos aus OpenStreetMap anzeigt. Das Gadget erleichtert das Erstellen neuer Links zwischen Wikipedia und OpenStreetMap, indem es das Ergebnis einer OSM-Suche nach Wikidata sowie Titel anzeigt. Wenn nichts gefunden wird, kann man einen direkten Link zu OSM verwenden, um das Objekt manuell zu suchen. (Mediawiki, 2017)

Das Tool fördert und erleichtert lediglich das setzen der Wikidata-Tags, nimmt jedoch keine Qualitätsprüfungen vor.

OSM Wikidata Explorer

Dieses Tool[18] ist dazu gedacht, die Distanz zwischen OSM und Wikidata-Koordinaten zu visualisieren. Grosse Abweichung könnten dabei ein Indiz für fehlerhafte Verknüpfungen sein. Es gibt gemäss dem Projekt vier Gründe für eine grosse Abweichung:

- Valide: Große natürliche Gegebenheiten wie Ozeane, Kontinente, Gebiete, die nicht an einem bestimmten Punkt liegen, können weit voneinander entfernt sein.
- Unpassendes Tagging auf OSM: Ein Wikidata-Tag wurde zu mehreren OSM-Objekten hinzugefügt, die ein einzelnes Objekt beschreiben, wie z.B. die Ways (Abschnitte) eines Flusses anstelle der Fluss-Relation.
- Falsche Verknüpfung: Ein Wikidata-Tag eines ähnlich benannten Objektes wurde fälschlicherweise zu OSM verknüpft.
- Falsche Koordinaten auf Wikidata: Das Wikidata-Item hat falsche und ungenau Koordinaten eingetragen.

Der angegeben Link[19] funktioniert zur Zeit nicht und das Projekt wurde seit 2017 nicht mehr aktualisiert (OpenStreetMap, 2022i). Grundsätzlich verfolgt es aber einen ähnlichen Ansatz und würde der Qualitätsverbesserung dienen, wenn es aktiv genutzt würde.

2.2 Requirements

Ausgehend von der Aufgabenstellung und des Systemkontext, wurden folgende Anforderungen an das Tool definiert. Die Requirements werden im Kapitel 5 verifiziert.

2.2.1 Funktionale Anforderungen

Folgende funktionalen Anforderungen soll das Tool erfüllen. Ausser der einen optionalen Anforderung, sind dies als Must-Have definiert.

Fehler in den Wikidata-Tags finden

Das Tool soll mögliche Fehler in den existierenden Wikidata-Tags finden.

Vorschläge zur Korrektur generieren

Das Tool soll den Benutzer beim Lösen der Issues unterstützen und sinnvolle Vorschläge zur Korrektur der Wikidata-Tags generieren.

Gefundene Fehler melden

Die gefunden Fehler sollen in einem Reporting in ein bestehendes Monitoring-Tool oder Fehlerdatenbank gemeldet werden. Vorzugsweise ist dabei das existierende und weitverbreitete Quality Assurance Tool 'Osmose' zu wählen.

Monitoring und Logging

Ein einfaches Monitoring soll helfen, die korrekte Funktionalität des Tools zu überprüfen, mögliche Fehlfunktionen frühzeitig zu erkennen und einen reibungslosen Betrieb zu garantieren. Ein Graphical User Interface (GUI) ist nicht vorgesehen.

Matching-Vorschläge machen (optional)

Als optionales Ziel können Matching-Vorschlage für noch nicht existierende oder falsch verlinkte Wikidata Tags gemacht werden.

2.2.2 Nichtfunktionale Anforderungen

Die nicht-funktionalen Anforderungen stellen den reibungslosen Betrieb sowie die Qualität der Checks und des Tools sicher, sodass das Tool produktive eingesetzt und weiterentwickelt werden kann.

Functional Suitability (Accuracy)

Da die gefundenen Fehler in den Verknüpfungen von Hand behoben werden müssen, ist es wichtig nicht zu viele False Positives zu generieren. Es ist eine True Positive Rate von über 90% anzustreben. Dabei wird in Kauf genommen, dass es False Negatives gibt.

Performance (Time-Behavoir and Resource Utilisation)

Bezüglich der Performance des Systems gibt es Vorgaben bezüglich der Laufzeit sowie der Ressourcen.

Das System soll mit den grossen Datenmengen der Dumps zurechtkommen, welche komprimiert für OpenStreetMap 64 GB und für Wikidata 70 GB betragen und stetig anwachsen. Da diese Dumps wöchentlich erscheinen, soll das System eine Laufzeit von maximal einer Woche auf dem von der OST zur Verfügung gestellten Server nicht überschreiten.

Reliability (Fault Tolerance)

Das System soll so erstellt werden, dass es bei einem einzelnen Fehler nicht abbricht, sondern versucht, die Tätigkeiten weiterzuführen. Es soll vor allem verhindert werden, dass einzelne fehlerhafte Daten oder Anomalien in den Dumps während dem Import, dem Überprüfen der Verknüpfungen oder beim Meldeprozess zum kompletten Abbruch des Programms führen. Stattdessen soll das Programm den Fehler isolieren, protokollieren und versuchen, das Programm fortzusetzen.

Maintainability

Da das Tool ein dauerhafter Teil der Infrastruktur zur Qualitätssicherung von OpenStreetMap werden soll, ergeben sich mehrere Ansprüche an die Maintainability.

So soll der Code eine genügend hohe Qualität aufweisen, dass er von anderen Entwicklern verstanden und gewartet werden kann.

Das Tool soll ausserdem Unit Tests mit einer Testabdeckung von 80% aufweisen. Fehlermeldungen und Logs sollen helfen, Programmierfehler oder Anomalien frühzeitig zu erkennen und zu lösen.

Ein modularer Aufbau der Software soll die Wartung und Weiterentwicklung des Codes ermöglichen. Architektur und Software sollen ausreichend dokumentiert werden, um anderen Entwicklern einen Überblick zu verschaffen und den Einstieg zu erleichtern.

Operability

Der Betrieb soll in einem Docker Container erfolgen und somit systemunabhängig verwendet werden können. Das System läuft und beendet sich selbständig, es sollen während des Prozesses keine Interaktion mit dem Tool nötig sein. Die Konfiguration soll flexibel sein, jedoch so gestaltet werden, dass sinnvolle Default-Werte hinterlegt sind und es somit im Normalfall ohne oder mit minimaler Konfiguration auskommt. Es ist kein GUI vorgesehen. Ein detailliertes und strukturiertes Handbuch soll helfen, das Tool schnell in Betrieb zu nehmen sowie es warten und weiterentwickeln zu können.

KAPITEL 3

Design

In diesem Kapitel werden die Überlegungen und Entscheidungen zum Softwaredesign beschrieben. Die Grundlage dafür bilden die in der Analyse definierten Anforderungen. Zuerst kommt ein Überblick über die Applikation und den Datenfluss, danach die Entscheidungen, welche zu dieser Architektur geführt haben, sowie Details zu relevanten Bereichen.

Relevante Design-Entscheidungen sind gemäss der Y-Methode nach Zimmermann (2020) dokumentiert, welche im Kapitel Projektmanagement (Unterabschnitt 7.1.2) näher erläutert wird.

3.1 Übersicht und Datenfluss

Der Abbildung 3.1 ist der Ablauf und Datenfluss der Applikation zu entnehmen. Es beginnt mit dem Herunterladen der Rohdaten für OSM, WD sowie Redirect aus dem Internet. Nachdem die Downloads abgeschlossen sind, startet das Programm mit dem Import der Dumps in die jeweilige Datenbank. Sind alle Daten importiert, kann der Checker beginnen. Ihm werden die einzelnen Datenobjekte der OSM Datenbank übergeben. Der Checker hat Zugriff auf die WD- und Redirects-Datenbank, um Informationen über die verlinkten Items zu erhalten und diese zu prüfen. Findet der *Checker* Fehler, so werden diese dem *Reporter* übergeben. Der *Reporter* aggregiert alle Issues, reichert sie mit Osmose-spezifischen Informationen an und erstellt eine XML-Datei, welche anschliessend an das Osmose-Frontend gesendet werden kann.

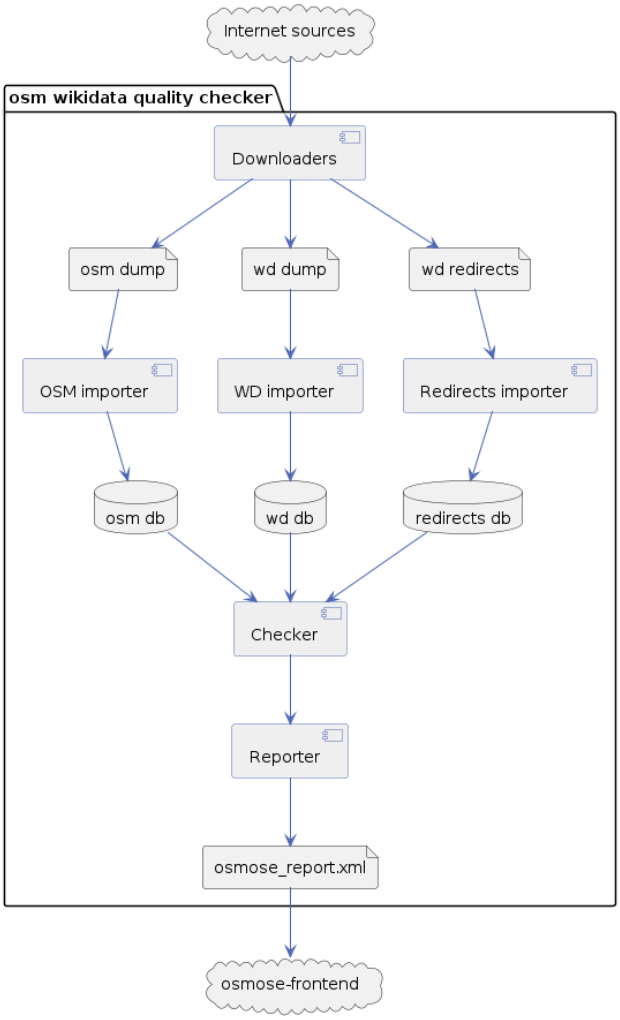


Abbildung 3.1: Datenfluss in der Applikation

3.2 Architektur

Die Abbildung 3.2 zeigt eine vereinfachte Darstellung des Tool und seiner Komponenten. Eine Komponente kann dabei aus mehreren Source-Files bestehen. Die Pfeile zeigen die Abhängigkeiten untereinander auf. Die Architektur verfolgt das Ziel, die Verantwortlichkeiten der Module klar zu unterteilen und feste Abhängigkeiten zwischen den Modulen zu verhindern. Zudem sollen die Komponenten, vor allem die Checks, einfach erweiterbar sein. In den folgenden

Unterkapiteln sind die einzelnen Komponenten genauer beschrieben.

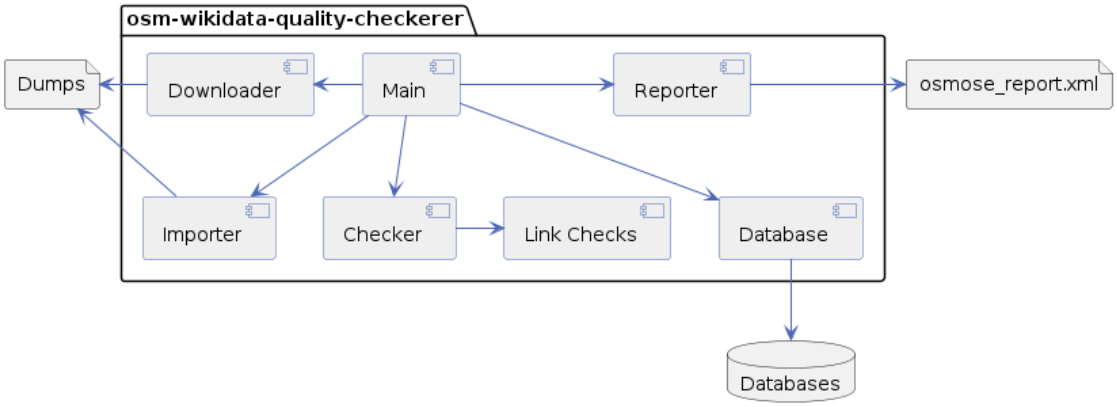


Abbildung 3.2: Komponentendiagramm und Abhängigkeiten (vereinfachte Darstellung)

3.2.1 Main

Das Hauptmodul verbindet alle anderen Komponenten und steuert den Ablauf des Programms. Es lädt die Environment-Variablen aus der Konfiguration (siehe Abschnitt 3.6) und übergibt diese in die anderen Bereiche der Applikation. Aufrufe zwischen den Modulen sind über Interface abstrahiert und werden vom Main übergeben. Mit dieser Dependency Injection wird verhindert, dass die einzelnen Komponenten direkte Abhängigkeiten untereinander haben.

3.2.2 Downloader

Der Downloader ladet die Dumps von den definierten Adressen herunter und speichert diese im konfigurierten Verzeichnis ausserhalb des Dockers. Die einzelnen Downloads werden parallel in separaten Prozessen ausgeführt.

Ob und von welcher Quelle die Downloads heruntergeladen werden sollen, ist einstellbar (siehe Anhang B). Diese Entscheidung wurde auf der Grundlage der folgenden Überlegungen getroffen:

Y

> Im Zusammenhang mit dem Download der verschiedenen Daten-Dumps, mit der Anforderung einer Gesamtlösung und gleichzeitiger Flexibilität, entschieden wir uns für den Einbau des Downloads-Mechanismus in das Tool selbst und verwerfen die Annahme, dass die Dump-Files vor dem Start des Tools extern bereitgestellt werden müssen. So erreichen wir die 'All-In-One' Anforderung des Tools, bieten jedoch gleichzeitig die Flexibilität, den Download auszuschalten, falls die Files extern bereits vorhanden sind. Dabei wird in Kauf genommen, dass ein leicht höherer Entwicklungsaufwand entsteht.

3.2.3 Importer

Nachdem die Dumps heruntergeladen wurden, kümmern sich die Importer um das Einlesen der Dumps in die Datenbanken. Die drei Importer für OSM, WD und die Redirects sind sehr ähnlich aufgebaut und können unabhängig voneinander verwendet werden. Sie kommunizieren über ein einheitliches Datenbank-Interface mit den jeweiligen Datenbanken.

Es gäbe auch die Möglichkeit, eine komplette Kopie der Datenbank zu unterhalten, in dem die regelmässig erscheinenden Changesets eingelesen werden. Da die vollständigen und dekomprimierten Datensätze von OSM und WD mit je ca. 1.5TB jedoch sehr gross sind, wurde dieser Ansatz mit dem nachfolgenden Entscheid verworfen:

Y Im Zusammenhang mit Verarbeitung der Dumps, mit der Anforderung der Performance (Time-Behavoir and Resource Utilisation), entschieden wir uns für das Erstellen von eigenen Datenbanken, in welchen nur die benötigten Daten importiert werden und verwerfen den Ansatz, eine komplette Kopie der Datenbanken zu unterhalten. So können die Daten in genügend hoher Geschwindigkeit mit zugleich minimalen Speicherplatzanforderungen verarbeitet werden. Dabei wird in Kauf genommen, dass ein eigenes Datenmodell entwickelt werden muss und falls weitere Datenfelder aus den Dumps benötigt werden, dieses angepasst werden muss.

Im Prototyp wurde die Möglichkeit getestet, die OSM Objekte direkt nach dem Laden aus dem Dump zu prüfen, anstatt einen Index aufzubauen. Theoretisch ist auf OSM Objekte im Gegensatz zu WD kein Random Access nötig und die Reihenfolge spielt ebenfalls keine Rolle.

Y Im Zusammenhang mit dem Iterieren über sämtlicher OSM Objekte, um diese überprüfen zu können, entschieden wir uns für das Erstellen einer temporären Datenbank und verwerfen die Möglichkeit, die Objekte direkt beim Durchlaufen des Dumps zu überprüfen. So erhält man Random Access auf die OSM Objekte, was für Analysen, Fehlersuche in Entwicklung und Betrieb wertvoll ist. Dabei wird in Kauf genommen, dass Speicherplatz für die Datenbank benötigt wird.

3.2.4 Datenbank

Die Database-Module bieten einen abstrahierten Zugriff auf die drei Datenbanken für OSM, WD und Redirects und die darin enthaltenen Objekte. So wird der Rest des Programms von der Datenbanktechnologie unabhängig gemacht und der direkte Datenbankzugriff verhindert. Um den Datenbankzugriff sicherer zu machen, wird ein OR-Mapper (siehe Unterabschnitt 4.4.4) verwendet.

Y Im Zusammenhang mit dem Einfügen und Auslesen der Datenbankobjekte, mit der Anforderung an Reliability und Maintainablity, entschieden wir uns für den Einsatz eines OR-Mappers und verwerfen das direkte Ansprechen der Datenbank über SQL. So erreichen wir eine höhere Fehlersicherheit beim Entwickeln und eine grössere Unabhängigkeit von der verwendeten Datenbanktechnologie. Dabei wird in Kauf genommen, dass ein geringer Overhead und eine Abhängigkeit zu dem verwendeten OR-Mapper entstehen.

3.2.5 Checker

Der Checker beinhaltet die Logik, um die Verknüpfungen von OSM zu Wikidata zu überprüfen. Dazu bekommt der Checker OSM-Objekte sowie Zugriff auf die WD- und Redirects-Datenbank vom Main übergeben und führt die in Abschnitt 3.3 beschriebenen Checks aus. Um die Entkopplung zu garantieren, werden gefundene Fehler an das Main Modul zurückgegeben, welches diese an einen Reporter weitergeben kann.

3.2.6 Reporter

Ein Reporter bekommt laufend Issues gemeldet und erstellt aus diesen einen Report. Wie in Abbildung 3.3 zu sehen ist, besitzen alle Reports ein einheitlichen Interface. So können diese kombiniert und ausgetauscht werden.

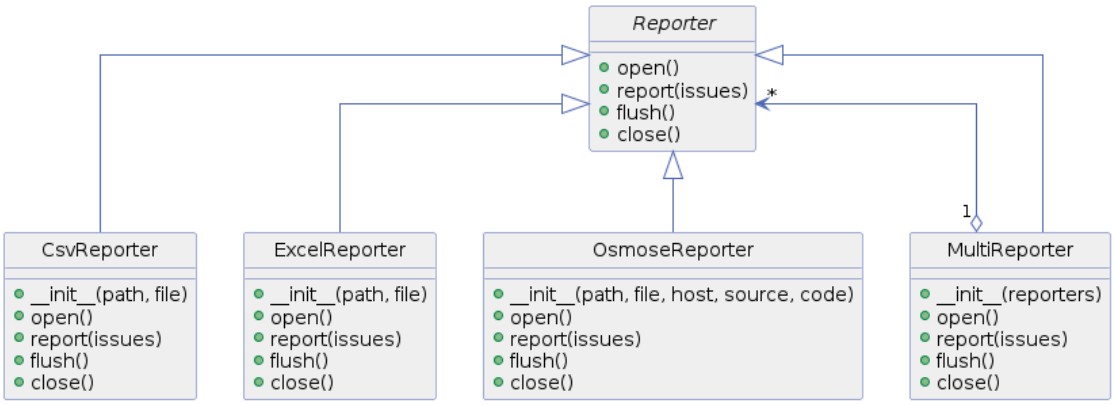


Abbildung 3.3: Reporter Klassen Diagramm

Im Falle des Osmose-Reporters erstellt dieser eine XML-Datei und sendet sie am Schluss an das Osmose-Frontend. Ein Beispiel eines solchen Osmose-Report ist in Anhang F zu finden. Der Excel- und der CSV-Reporter produzieren eine Datei im entsprechende Format mit allen Issues. Diese sind für die Optimierung der Checks sowie Fehlersuche und Statistikzwecke im Betrieb sinnvoll. Der Multi-Reporter implementiert das Composite-Pattern. So ist es möglich, mehrere Reports gleichzeitig und über dieselbe Schnittstelle zu generieren (Gamma, Helm, Johnson, Johnson & Vlissides, 1995).

3.3 Checks

Im folgenden Kapitel sind die einzelnen Checks beschrieben, welche die Verknüpfungen zwischen OSM und WD prüfen.

3.3.1 Invalid WD-Format

Als erster und wichtigster Check wird geprüft, ob in einem Wikidata-Feld ein gültiger Wert steht. Ein Wikidata Item muss mit dem Grossbuchstaben Q, gefolgt von einer Nummer angegeben werden. Mehrere solcher Q-Id's müssen mit Semikolon getrennt werden. Tabelle 3.1 zeigt ein Beispiel aus der Praxis von gültigen und ungültigen Werten.

Wert	Kommentar
`Q10641`	gültig
`Q692884;Q171518`	gültig
`;;Q314;Q2989;;`	ungültig (unnötige Trennzeichen)
`Q94639|Q2492340`	ungültig (ungültiges Trennzeichen)
`eQ86666189`	ungültig (ungültiges Format)
`https://www.wikidata.org...`	ungültig (mit URL)
`Kirschbaum`	ungültig (Freitext)
`L614072`	gültig, jedoch nicht geprüft (Lexeme)

Tabelle 3.1: Beispiel von gültigen und ungültigen Formaten für Wikidata Values

Im Laufe der Entwicklung hat sich gezeigt, dass teilweise auch Lexeme[20] verwendet werden. Diese sind zwar nicht explizit von der Community als erlaubt angegeben (OpenStreetMap, 2022c), jedoch für Etymologie-Angaben sinnvoll. Daher wurde beschlossen, diese im Zusammenhang mit

einem `name:etymologie:wikidata` Tag zu erlauben. Jedoch werden keine weiteren Checks durchgeführt und der Eintrag somit von nachfolgenden Checks ignoriert. Im Zusammenhang mit einem nachgestellten `...:missing` oder `...:fixme` Tag wird teilweise Freitext eingegeben. Die Verwendung dieser Postfixes beim Wikidata-Tag ist ebenfalls nicht explizit spezifiziert. Es wurde jedoch darauf verzichtet, diese als Fehler zu melden, sodass solche Werte ebenfalls ignoriert werden.

3.3.2 Not Existing WD-Item

Bei diesem Check wird geprüft, ob ein Wikidata-Item zur angegeben Q-Id existiert.

Dabei wird geschaut, ob die Q-Id in der WD-Datenbank vorhanden ist oder nicht. Wegen der zeitlichen Differenz zwischen den beiden Datenbank-Dumps, kann es gelegentlich vorkommen, dass ein verlinktes Item noch nicht im Wikidata-Dump enthalten ist. Deshalb erfolgt zusätzlich eine Abfrage über die Wikidata-API (siehe Unterabschnitt 2.1.3). Dieses Verhalten wird beim Redirected WD-Item Check, in Unterabschnitt 3.3.3, genauer erklärt.

3.3.3 Redirected WD-Item

Dieser Check ist eng mit dem vorhergehenden Not Existing WD-Item Check (Unterabschnitt 3.3.2) verknüpft. Falls eine Q-Id nicht in der WD-DB gefunden wurde, wird bei diesem Check geprüft, ob es sich um eine Weiterleitung handelt.

Bei Wikidata Items kann es vorkommen, dass ein und dasselbe Ding doppelt erfasst wird. Erkennt die Community einen solchen Fall, werden die Artikel zusammengeführt. Einer der beiden Einträge wird als Weiterleitung auf den Haupteintrag betrachtet. Gemäss OpenStreetMap (2022h) sollen keine Redirected-Id's verlinkt werden, nur der Haupteintrag ist korrekt. Deshalb wird geprüft, ob die Q-Id in der Redirects-DB vorkommt und es sich somit um ein Redirect handelt.

Auch bei den Redirects-Rohdaten können, wegen der Zeitdifferenz zu den Dumps, Einträge noch nicht vorhanden sein. Weshalb in einem letzten Schritt die Wikidata-API (siehe Unterabschnitt 2.1.3) angesprochen wird. Diese liefert entweder das angefragte Item, ein weitergeleitetes Item oder einen Not-Found Fehler zurück. Wurde über die API ein Item entdeckt, welches existiert aber nicht im Dump vorkommt, wird es von den nachfolgende Checks ignoriert. Spätestens beim nächsten Durchlauf der Applikation sollte das Item auch im Wikidata-Dump vorhanden sein und kann dann komplett geprüft werden. Abbildung 3.4 zeigt den kompletten Ablauf der zwei Checks.

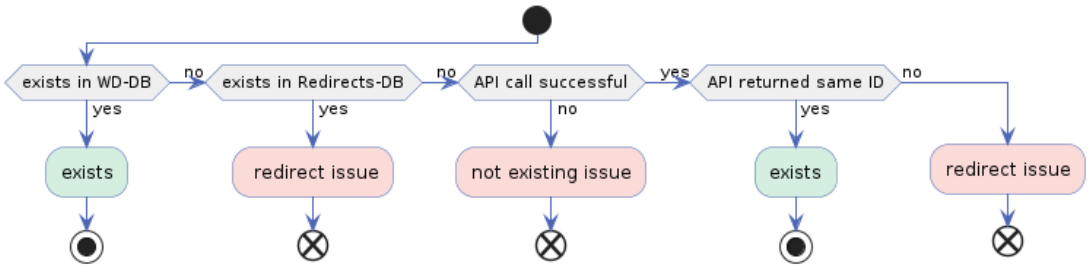


Abbildung 3.4: Zusammenspiel vom "Not existing" und "Redirected" Check

3.3.4 Unpermitted Instance

Dieser Check sucht nach Verknüpfungen auf Wikidata Items mit unerlaubten Kategorien.

Ein verlinktes Item muss direkt mit dem OSM-Objekt zusammenhängen und darf nicht auf Meta-Objekte oder Sammlungen zeigen. Unerlaubte Kategorien können anhand der im Claim *Instane*

of (P31) erfassten Werte erkannt werden. Die Tabelle 3.2 zeigt auf, nach welchen ungültigen Instanzen gesucht wird. So ist es zum Beispiel nicht erlaubt auf Wikidata-interne Objekte oder eine Listen zu verweisen (OpenStreetMap, 2022c).

Instanz	**Beschreibung gemäss Wikidata**
Wikimedia disambiguation page (Q4167410)	Seite mit Verweisen auf gleichlautende Lemmata mit verschiedenen Bedeutungen
Wikimedia category (Q4167836)	in Kombination mit dem Property *instance of* (P31) zu verwenden
Wikimedia list article (Q13406463)	Seite eines Wikimedia-Projektes, bei der es primär um eine Auflistung von etwas geht
list (Q12139612)	Sammlung von Informationen

Tabelle 3.2: Ungültige Instanzen von verlinkten Wikidata Items

Ein häufig anzutreffender Fall ist, bei `name:etymologie:wikidata` auf eine Disambiguation Page zu verweisen. Dabei ist es gerade bei der Namensherkunft relevant, auf den konkreten Eintrag zu verweisen. Ist also der "Washington Plaza" nach dem Präsidenten George Washington, dem Chemiker und Erfinder George Washington Carver oder der Hauptstadt Washington D.C. benannt? Eine Liste aller Washingtons dieser Welt beantwortet diese Frage nicht und der Eintrag macht deshalb keinen Sinn und wird als Fehler gemeldet. (OpenStreetMap, 2022b)

Der Check prüft die wichtigsten und häufigsten anzutreffenden Fehler und ist weiter ausbaubar, vor allem mit Unterklassen der betreffenden Oberklassen. Es darf dabei nur nach direkten Instanzen gesucht werden und nicht in der Hierarchie nach oben, da sich in der Datenstruktur von Wikidata bei höheren Abstraktionslevels irgendwann alles von Wikidata-Internals oder einer Liste ableitet.

3.3.5 Living Organism

Der Check prüft, ob es erlaubt ist, für den Wikidata-Tag einen Link auf einem Wikidata-Item anzugeben, der einen lebenden Organismus (Mensch, Tier, Pflanze, Pilz) bezeichnet.

Idee

In OSM existieren nur geografische Objekte, so ist es in den meisten Fällen nicht möglich, auf etwas lebendiges zu verweisen. Für gewisse Sekundär-Tags ist eine Verknüpfung zu einem Living Organism jedoch sinnvoll. Diese sind in Tabelle 3.3 zu sehen.

Nicht erlaubt (explizit geprüft)	**Beispiele erlaubt (nicht abschliessend)**
`wikidata`	`subject:wikidata`
`brand:wikidata`	`buried:wikidata`
`operator:wikidata`	`artist:wikidata`
`network:wikidata`	`species:wikidata`
`flag:wikidata`	`...`

Tabelle 3.3: Ungültige Tags für lebendige Organismen

Beispiel

Ein sehr häufiger anzutreffender Fehler ist, dass bei einer Statue die abgebildete Person mit dem Primär-Tag `wikidata` anstatt mit dem Sekundär-Tag `subject:wikidata` verlinkt ist. Beim Primär-Tag

darf nur ein Artikel über die Statue selbst angegeben werden. Die Abbildung 3.5 zeigt, wie eine Statue korrekt verlinkt werden muss.

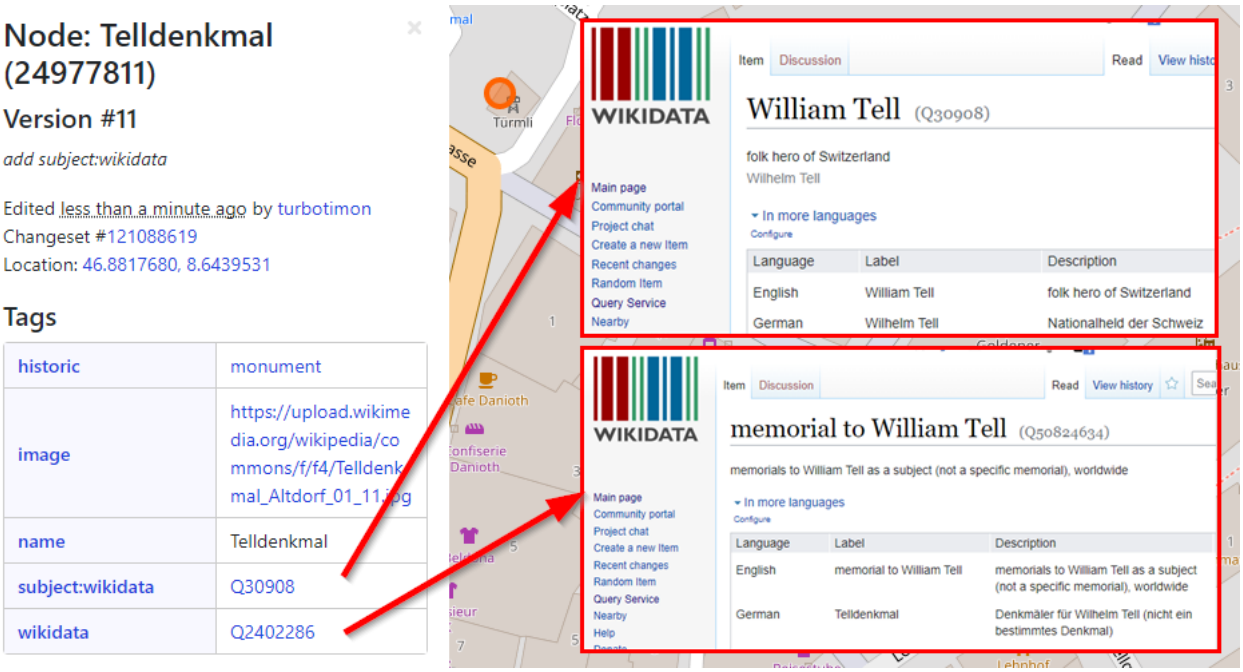


Abbildung 3.5: Beispiel einer korrekt verlinkten Statue

Selbiges gilt für Grabstätten, wo für die Person `buried:wikidata` verwendet werden sollte. Ein weiteres Beispiel ist, wenn bei einem Baum direkt auf die Sorte verlinkt wird, anstatt den `species:wikidata`-Tag zu benutzten.

Aktueller Stand

Der Test prüft, ob das verlinkte Item eine Instanz von *living organism class* (Q21871294) ist, oder von dieser ableitet. So werden auch Unterklassen und somit sämtliche lebendigen Dinge erkannt. Um neu hinzukommenden Tags nicht automatisch und eventuell fälschlicherweise als Fehler zu erkennen, wird nur auf die nicht erlaubten Tags geprüft. Bei allen anderen sind Lebewesen erlaubt. Der Check ist ziemlich eindeutig, False Positives treten gelegentlich auf, falls ein Wikidata-Item einen unkorrekten Taxonomie-Claim hinterlegt hat.

3.3.6 Primary Tag Claim Mismatch

Prüft, ob die beim OSM-Objekt hinterlegten Tags mit der Kategorie des primär verlinkten Wikidata-Eintrag zusammenpasst.

Idee

Die Blohm + Voss Schiffswerft im Hamburger Hafen (Way 19059511) besitzt den Tag `industrial=shipyard`. Das verlinkte Wikidata Item Q218715 sollte demnach eine Instanz einer *Schiffswerft* (Q190928) sein. Da es sich offensichtlich um eine Region handelt, auf der Industrielle Arbeiten verrichtet werden, darf der Hafen auch auf eine *Industrieregion* (Q6027980) oder ein *Industriegebäude* (Q12144897) zeigen. Dies ist als Fallback-Regel zu verstehen und gilt für sämtliche Industrial-Tags (`industrial=*`). Sollte auch die Fallback-Regel gebrochen werden, so

ist die Verlinkung mit grosser Wahrscheinlichkeit falsch oder es müsste ein sekundärer Link wie `operator:wikidata` oder `owner:wikidata` verwendet werden. Tabelle 3.4 zeigt einige Beispiele von spezifischen und generellen Regeln für den Tag `industrial=*`.

Tag	Wikidata Instance Check
`industrial=*`	*Industrieregion* (Q6027980), *Industriegebäude* (Q12144897)
`industrial=shipyard`	*Schiffswertft* (Q190928)
`industrial=oil`	*Ölindustrie* (Q862571), *Öl-Feld* (Q211748), *Raffinerie* (Q1867977), *Tanklager* (Q592856)
`industrial=port`	*Hafen* (Q44782), *Hafen* (Q283202), *Dock* (Q124282)

Tabelle 3.4: Primary Tag Check Regelbeispiele für Tag `industrial`

Ein initiales Set an Regeln kann aus Wikidata generiert werden. Das Wikidata Property *OpenStreetMap tag or key* (P1282) gibt an, welche OSM Tags wie gesetzt sein müssen, damit eine Instanz der aktuellen Klasse verlinkt werden darf. Mit der in Listing 3.1 gezeigten SPARQL-Suche können alle Items gefunden werden, welche einen solchen Claim erfasst haben.

```
SELECT DISTINCT ?item ?itemLabel WHERE {
  SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE]". }
  {
    SELECT DISTINCT ?item WHERE {
      ?item p:P1282 ?statement0.
      ?statement0 (ps:P1282) _:anyValueP1282.
    }
  }
}
```

Listing 3.1: SPARQL-Query um alle Items mit P1282-Claim zu finden

Probleme

Die aus dem Claim *OpenStreetMap tag or key* (P1282) resultierenden Regeln ergeben ein gutes Fundament. Sie sind aber zu streng und müssen von Hand stark überarbeitet werden. So müsste im Falle von `industrial=oil` das Wikidata-Item eine Instanz oder Subklasse von *Ölindustrie* (Q862571) sein. Auf OSM ist es aber ebenso etabliert und korrekt, wenn ein z.B. ein *Öl-Feld* (Q211748) oder ein *Tanklager* (Q592856) das Tag `industrial=oil` trägt. In Tabelle 3.4 ist deshalb ersichtlich, dass auch Öl-Felder oder Tanklager erlaubt sind.

Desweiteren ist dieser Check nicht bei jedem OSM-Tag sinnvoll. Der Tag `building` kommt beispielsweise viel öfter vor als `industrial`. Bei diesem ist es jedoch schwierig zu definieren, was alles erlaubt ist. Gemäss OpenStreetMap (2022a) darf bei Building der ursprüngliche Verwendungszweck eingetragen werden, auch wenn das Gebäude nicht mehr so genutzt wird. Zum Beispiel ist das *Altes Schulhaus Rüttenen* (Way 52773265) immer noch ein `building=school`, auch wenn dort nicht mehr unterrichtet wird.

Ausserdem ist bei 80%[21] der Fälle nur `building=yes` eingetragen, was einen sinnvollen Check nahezu verunmöglicht. Ein solches Beispiel ist das Hotel *Revier Mountain Lodge* (Way 690768009). Das verknüpfte Wikidata-Item (Q111409850) ist ein *Hotel* (Q27686). Hier wäre es jedoch theoretisch möglich, über den zusätzlichen Zweck `tourism=hotel`, eine Prüfung vorzunehmen.

Aktueller Stand

Das Erstellen des Regelsets, wie auch die Validierung davon benötigen sehr viel Zeit. Aus diesem Grund wählte man einen kleineren Umfang für den Check, damit der Nutzen des Checks trotz hohem Zeit-Investment aufgezeigt werden kann. Aktuell werden die in Tabelle 3.5 aufgelisteten Tags vom Checker überprüft. Tabelle 3.6 zeigt Fehler-Beispiele des Primary Tag Checker.

Geprüfte Tags	
`place=village`	`place=neighbourhood`
`place=country`	`industrial=shipyard`
`place=county`	`industrial=port`
`place=suburb`	`industrial=harbour`
`place=town`	`industrial=natural_gas`
`place=hamlet`	`industrial=gas`
`place=city`	`amenity=school`

Tabelle 3.5: Tags, welche durch den Primary Tag Checker überprüft werden

OSM Objekt	Bemerkung	Konsequenz
Itzalle / Izal (Node 3590725237)	Dieses Dorf zeigt auf den Wikidata-Eintrag *shadow* (Q160020) (Schatten).	**True Positive:** Verknüpfung ist klar falsch (die baskische Übersetzung von ”Schatten” ist anscheinend ”Izal”).
Maniów (Node 692614097)	Dieser Weiler zeigt auf den Wikidata-Eintrag *Maniów, Podkarpackie Voivodeship* (Q5191478) was korrekt scheint. Dieser ist jedoch als Kategorie markiert, also als Meta-Objekt und nicht als physisches Dorf.	**False Positive:** Aufgrund des inkorrekten Claim's auf Wikipedia (Datenqualität)
Schule (Node 198404677)	Der Node hat den Tag `amenity=school`, der Wikidata-Eintrag ist jedoch auf die Gemeinde *Monte Buey* (Q2506369).	**True Positive:** Nur direkte Verknüpfungen sind erlaubt. Die Gemeinde müsste mit `is_in:wikidata` oder ähnlich hinterlegt werden.

Tabelle 3.6: Beispiel-Fehler vom Primary Tag Checks

3.3.7 Secondary Tag Claim Mismatch

Prüft, ob die im OSM-Objekt beim Secondary Wikidata-Tag hinterlegten Einträge die Anforderungen an den sekundären Link-Typ erfüllen.

Idee

Auf OSM ist es möglich, sekundäre Wikidata Verknüpfungen zu erstellen wie zum Beispiel `brand:wikidata=Q312` oder `operator:wikidata=Q83835`. Die dabei hinterlegten Wikidata Items können darauf überprüft werden, ob sich hinter dem als `brand:wikidata` verlinkten Artikel tatsächlich eine Instanz einer *Marke* (Q431289) befindet.

Probleme

Die OSM Community gibt in ihren Richtlinien vor, welche Wikidata Instanzen für welche sekundären Tags erlaubt sind (OpenStreetMap, 2022c). Diese Einschränkungen sind aber zu spezifisch und die Realität ist komplexer. Als Beispiel ist die Einschränkung bei `brand:wikidata` rein auf *Marken* (Q431289) eine zu strikte Anforderung, *Handeslmarken* (Q167270) oder *Organisationen* (Q43229) sind ebenfalls sinnvoll und müssen deshalb auch erlaubt werden.

Zudem existieren diverse Zielkonflikte. So sollten bei `network:wikidata` gemäss OpenStreetMap (2022c) nur Items mit einem Bezug auf Netzwerke des öffentlichen Verkehrs verlinkt werden. Nur wird dieser Tag auch oft für E-Auto Ladenetze verwendet. Dies scheint von der Community aktuell akzeptiert zu werden. Solche häufig vorkommenden und allgemein akzeptierten Fälle müssen beachtet werden.

Aktueller Stand

Tabelle 3.7 zeigt Beispiele implementierter Regeln auf verschiedenen sekundären Tags. Total gibt es 18 solcher Regeln, welche eine hohe Treffsicherheit bieten.

Tag	Wikidata Instance Check
`brand:wikidata=*`	*Marke* (Q431289), *Handelsmarke* (Q167270), *Organisationen* (Q43229)
`architect:wikidata=*`	*Mensch* (Q5), *Gruppe von Menschen* (Q16334295), *Personenvereinigung* (Q783794), *Architekturbüro* (Q4387609), *Geschäftsbetrieb* (Q4830453)
`network:wikidata=*`	*Organisationen* (Q43229), *Verkehrsnetz* (Q924286), *Verkehrsinfrastruktur* (Q376799), *Buslinie* (Q3240003), *Marke* (Q431289), *Netzwerk* (Q1900326)

Tabelle 3.7: Secondary Tag Check Regelbeispiele

Tabelle 3.8 zeigt Beispiele, welche mit diesem Check erkannt werden.

OSM Objekt	Bemerkung	Konsequenz
Baum (Node 9132772093)	Hat in `species:wikidata` eine Verknüfpung zu *Palm* (Q1078850). Es handelt sich jedoch nicht um die Baumart *Palme*, sondern um Elektronikgeräte des Herstellers *Palm*.	**True Positive:** Verknüpfung ist falsch
Baum (Node 6807334009)	Hat in `species:wikidata` eine Verknüfpung zu *palm* (Q2001588). Es handelt sich jedoch nicht um die Baumart *Palme*, sondern um den Körperteil *Handfläche (en:palm)* .	**True Positive:** Verknüpfung ist falsch
Parkplatz (Way 1026301052)	Hat in `brand:wikidata` eine Verknüfpung zu *Behindertenparkplatz* (Q814499).	**True Positive:** Verknüpfung ist falsch, ein Behindertenparkplatz ist keine Marke.

Tabelle 3.8: Beispiel-Fehler vom Secondary Tag Checks

3.3.8 Place Mismatch

Der Check kontrolliert, ob Ortschaften auf OSM mit dem korrekten Wikidata Eintrag verlinkt sind, in dem eine Punketzahl über gegebene Merkmale berechnet wird.

Idee

Ein bedeutender Tag auf OpenStreetMap mit fast 8 Millionen Verwendungen[22] ist `place`. Mit diesem werden jegliche Arten von Örtlichkeiten gekennzeichnet. Als Werte können sowohl menschliche Siedlungen wie `place=city` wie auch geografische Objekte `place=continent` eingegeben werden.

Der Check vergleicht die in Tabelle 3.9 aufgelisteten Merkmale von OSM und Wikidata und errechnet daraus eine Punktezahl, anhand deren entschieden wird, ob die Verknüpfung valide ist. Isoliert betrachtet, sind die Merkmale für einen Entscheid ungenügend. In Kombination liefern sie jedoch starke Indizien. Anhand der Gewichtung der einzelnen Scores, wie auch dem Fehler-Threshold beim Total Score, kann die Sensitivität des Tests eingestellt werden.

OSM Merkmale	Wikidata Merkmal	Vergleichsmethode
Name-Tag's wie `name`, `old_name`, `loc_name` ...	Alle Bezeichnungen (Labels)	Bester String-Match mittels des Levenshtein-Algorithmus
Punkt-Koordinaten (Node) oder Polygon (Areas, Ways)	Alle Koordinaten (*coordinate location* (P625))	Mininmale Distanz mittels der Haversine-Formel
Postleitzhal `postal_code`	Postleitzahlen (*postal code* (P281))	Bester String-Match mittels des Levenshtein-Algorithmus

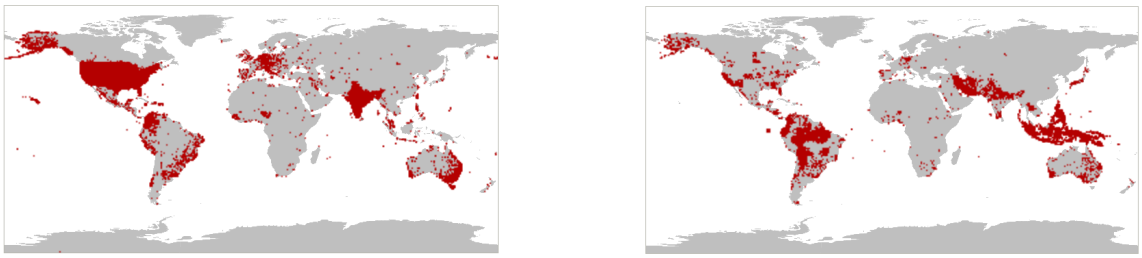
Tabelle 3.9: Verwendete Merkmale für Place Check

Probleme

Es existieren diverse Ortschaften, die wegen ungenauer oder fehlender Daten auf Wikidata schwer zu prüfen sind. Probleme sind unter anderem:

- Der Namensvergleich ist, wegen Präfixes oder Postfixes (Beispiel "La Palma" vs. "Palma") oder durch fehlende Übersetzungen (Beispiel OSM nur griechisch, Wikidata nur englisch), zu ungenau.
- Die Koordinaten auf Wikidata sind oft sehr ungenau und mehrere Kilometer vom eigentlichen Objekt entfernt. Deshalb ist eine grosse Toleranz notwendig.
- Das Merkmal Postleitzahl ist sowohl auf OpenStreetMap wie auch Wikidata selten vorhanden. Zudem ist das Format je nach Land verschieden, sodass nur ein String-Vergleich machbar ist.

Verschiedene andere Merkmale wie Adresse oder Region bez. Bundesland könnten helfen, den Test noch präziser zu machen. Jedoch sind diese, entweder in Wikidata oder auf OSM, spärlich gesetzt. Zusätzlich werden gewisse Tags von Land zu Land unterschiedlich verwendet. Eines von unzähligen solcher Beispiele ist in Abbildung 3.6 zu sehen.



(a) Verwendung von `addr:state`

(b) Verwendung von `is_in:state`

Abbildung 3.6: Regional unterschiedliche Angabe des Bundeslandes in OpenStreetMap

Aktueller Stand

Der Check konzentriert sich auf die in Tabelle 3.10 sichtbaren Örtlichkeiten. Einerseits werden Werte mit hoher Häufigkeit wie `village` oder `hamlet` unterstützt, andererseits Werte mit grossem Anteil an Wikidata-Tags und somit hoher Relevanz. So sind zum Beispiel Länder oder Grossstädte für die Allgemeinheit relevanter als Weiler und Flure. Insgesamt werden so über 70% der Place-Tags abgedeckt und rund 650'000 Objekte überprüft. Bei den nicht abgedeckten Tags handelt es sich um geografische Objekte wie `island`, `continent` und `ocean` oder um noch kleinere Siedlungsformen wie `farm` und `isolated_dwelling` (Kleinstsiedlung). Während Letztere eine tiefe Relevanz und eine sehr niedrige Wikidata-Tag Abdeckung haben, gelten für Erstere andere Parameter und sollten getrennt betrachtet werden.

Wert	**Häufigkeit**	**Wikidata-Tag's**	**Definition**
country	224 (0.0%)	223 (99.6%)	Staat
state	1'958 (0.0%)	1'305 (66.6%)	Bundesland
county	8'981 (0.1%)	2'633 (29.3%)	Verwaltungsbezirk
city	20'034 (0.3%)	18'833 (94.0%)	Grossstadt
town	120'405 (1.6%)	81'976 (68.1%)	Städtische Siedlung
suburb	138'121 (1.8%)	21'813 (15.8%)	Teil einer Stadt
village	1'486'183 (19.4%)	311'785 (21.0%)	Dorf
hamlet	1'759'434 (23.0%)	181'975 (10.3%)	Kleiner Ort oder Weiler
locality	1'642'944 (21.5%)	21'283 (1.3%)	Örtlichkeiten oder Flure (unbewohnt)
neighbourhood	376'859 (4.9%)	10'488 (2.8%)	Stadtviertel
Total	**5'555'143 (72.6%)**	**652'314 (11.7%)**	

Tabelle 3.10: Statistik über verwendeten Werte des Place-Tags

Die Sensitivität des Checks ist so eingestellt, dass möglichst wenig False Positives generiert werden. Somit wird aber in Kauf genommen, dass falsch verknüpfte Ortschaften nicht erkannt werden. False Positives treten besonders oft durch ungenau platzierten Koordinaten auf Wikidata auf. Die Tabelle 3.11 zeigt Beispiele solcher Objekte und deren Konsequenzen für den Check.

OSM Objekt	Bemerkung	Konsequenz
Zagrody (Node 31884938)	Das Wikidata Item bezieht sich auf einen anderen Weiler mit exakt gleichen Namen, circa 30km entfernt. Es ist keine Postleitzahl vorhanden.	**False Negative:** Der berechnete Score ist zu hoch um einen Fehler zu erkennen.
Archanes (Node 849370729)	Die Koordinaten des Wikidata Item zeigen 20km von der Stadt entfernt ins Meer. Die Namen matchen nur zu 83% (*Archanes* vs. *Ano Arhanes*). Es ist keine Postleitzahl vorhanden	**False Positive:** Die Verknüpfung ist korrekt, jedoch sind die Merkmale zu spärlich und ungenau, sodass ein Fehler generiert wird.
Słobódka (Node 31598517)	Das Wikidata Item zeigt auf eine 26km entfernte Siedlung mit exakt gleichem Namen. Die Postleitzahl ist verschieden.	**True Positive:** Nur anhand der unterschiedlichen Postleitzahl wird der Score genügend klein um diesen Fehler zu erkennen.

Tabelle 3.11: Beispiel von schwer zu erkennender Verknüpfungen

3.3.9 Very Large Distance

Der Check prüft die Differenz zwischen den auf OSM und Wikidata angegebenen Koordinaten.

Idee

Auf Wikidata können Koordinaten im Property *geographische Koordinaten* (P625) hinterlegt werden. Dabei ist es möglich, mehrere Koordinaten zu hinterlegen. Als Beispiel sind bei der *Ostschweizer Fachhochschule* (Q106411923) alle drei Standorte Rapperswil, St. Gallen und Buchs hinterlegt. Der Check prüft die minimale Distanz zu jedem der hinterlegten Koordinaten, die kleinste Distanz daraus wird für den Check verwendet.

Der Check ist nur beim Primär-Tag sinnvoll, da es zum Beispiel bei einer Angabe der Marke mit Angabe der Koordinaten des Hauptsitzes keinen Sinn macht, diese mit den Koordinaten einer Filiale zu vergleichen.

Probleme

Anfangs als simpler und treffsicherer Check gedacht, hat sich herausgestellt, dass es diverse valide Gründen für grosse Distanzen geben kann:

- Die Punkt-Koordinaten von grossen Gebilden ohne klar definierte Referenzpunkt können in OSM und Wikidata anders definiert sein. Beispielsweise liegt der Mittelpunkt von *Asien* (Q48) in Wikidata 500 km von Mittelpunkt auf OSM Node 36966065 auseinander.
- Die Datenqualität der Koordinaten auf Wikidata ist oft sehr schlecht. So zeigen die Koordinate des nordaustralischen Verwaltungsbezirks *Barkly* (Relation 8878329) ins 1'500 km entfernte Westaustralien.
- Eine Relation kann aus mehreren nicht verbunden Einzelobjekten (Areas) bestehen. In Wikidata ist der Mittelpunkt angegeben, der weder in noch genug nahe einer Area liegt. Bei der Inselgruppe *Kanarische Inseln* (Relation 349048) ist in Wikidata (Q5813) ein Punk im Meer angegeben.
- Einzelne Flussabschnitte (Way) sind mit dem Wikidata Item des gesamten Flusses verknüpft. Die Relation 36464 zeigt ein solches Beispiel. Da auf Wikidata nur einzelne Koordinate hinterlegt sind (die Quelle oder Mündung), kann es eine grosse Abweichung zum

einzelnen Flussabschnitt geben. Korrekterweise dürfte nur die Relation, welche den gesamten Fluss umfasst, mit dem Wikidata Item des Flusses verbunden werden. Jedoch kommt dieser Umstand so oft vor, dass er beachtet werden muss, um nicht massenhaft Fehler zu generieren. Auch bei Autobahnen kommt es vor, dass jeder Wegpunkt fälschlicherweise mit dem Item über die gesamte Strasse verbunden ist.

Diese Beobachtungen decken sich mit den im OSM Wikidata Explorers (Unterabschnitt 2.1.5) beschriebenen Fällen.

Aktueller Stand

Der Check schlägt nur bei ungewöhnlich hohen Abweichungen über 2000km an. Zusätzlich sind sämtliche Wasys und alle anderen Objekte mit Tag `water=river` sowie `place=ocean` und `place=continent` vom Check ausgeschlossen. So können die meisten Probleme umgangen werden und die genügende Treffsicherheit erreicht werden, auch wenn dabei eine vermutlich grössere Zahl von Fehler unerkannt bleibt. Die Tabelle 3.12 zeigt diesen Zielkonflikt bei der Wahl des Schwellenwertes. Der Check wird als letzter ausgeführt, da andere wie zum Beispiel der Place Checker (3.3.8) oder Primary Tag Checker (3.3.6) spezifischere Fehlermeldungen generieren können.

OSM Objekt	Bemerkung	Konsequenz
Triple R Ranch Airport (Node 9563826617)	Der Flughafen befindet sich Florida. Der Wikidata Eintrag zeigt jedoch auf den gleichnamigen Flughafen in Kalifornien (Node 369169540).	**True Positive:** Die grosse Distanz von 3'350 km führt zur korrekten Fehlererkennung. Andere Checks hätten wegen des gleichen Namens und Typs (Flughafen) Mühe den Fehler zu erkennen.
UNESCO Office (Node 2399342138)	Es handelt sich um eine Büro der Organisation UNESCO in Israel. Verlinkt ist die gesamte Organisation mit Koordinaten des Hauptsitzes. Es müsste korrekterweise der Key `operator:wikidata` verwendet werden, um das Problem zu beheben..	**True Positive:** Nur die grosse Distanz von 3'300 km gibt einen Hinweis auf einen Fehler. Weitere Tags, die bei der Fehlererkennung hilfreich wären, sind nicht vorhanden.
Embassy of Ecuador (Relation 4713850494)	Die ecuadorianische Botschaft in Mexico ist mit der in Costa Rica verknüpft	**False Negative:** Trotz der grossen Abweichung von 1'900 km liegt diese noch unter dem Schwellwert und wird deshalb nicht erkannt
Sathar island (Way 18761443)	Der Wikidata Eintrag ist zwar korrekt verklingt. Die Koordinaten der malaysischen Insel sind auf Wikidata jedoch 16'000 km entfernt im karibischen Meer vor Kolumbien platziert.	**False Positive:** Wegen der sehr ungenauen Koordinaten auf Wikidata lassen sich solche zu unrecht gemeldete Fehler schwer vermeiden.

Tabelle 3.12: Beispiele von korrekten und unkorrekten Very Large Distance Fehlern

3.3.10 Replaced-by

Der Check prüft, ob im verknüpften Wikidata Item das Feld *replaced by* (P1366) vorhanden ist und meldet entsprechende Fehler.

Idee

Mit dem Replaced-By Claim wird angegeben, wenn etwas durch etwas anderes ersetzt wurde. Die zu Grunde liegende Idee hinter diesem Check ist, dass auf OSM nur aktuelle Dinge kartographiert werden dürfen und keine historischen Daten (OpenStreetMap, 2022). Als Beispiel wurden im Jahre 2011 viele Gemeinden im Kanton Glarus zusammengelegt. So wird auf Wikidata die Gemeinde *Elm* (Q661631) mit einem *replaced by* (P1366) der neuen Gemeinde *Glarus Süd* (Q70695) vermerkt.

Ein weiteres gutes Beispiel ist die Fachhochschule *HSR* (Q1622219), welche durch die *OST* (Q106411923) ersetzt wurde. Es sollte also keine Verknüpfung mehr existieren, welche auf HSR zeigt, sondern der neue Eintrag verwendet werden.

Probleme

In der Realität ist die Situation jedoch sehr viel komplexer als ursprünglich gedacht. So ist *Elm* (Q661631) nicht nur eine *ehemalige Gemeinde* (Q19730508) sondern auch ein *Dorf* (Q532). Das Dorf Elm existiert noch und soll auch so verlinkt werden dürfen. Der Check müsste also in der Lage sein, solche Feinheiten zu unterscheiden, falls sie auf Wikidata korrekt erfasst sind.

Der Check ist auch nicht für alle Secondary-Wikidata-Tags sinnvoll. Zum Beispiel wird bei der Statue *Dom Pedro I* (Node 7706363320) mit dem Tag `subject:wikidata=Q939` die abgebildete Person verlinkt. Hatte diese ein politische Amt inne, wird mit dem Replaced-By Attribut sein Nachfolger angegeben. In diesem Fall darf jedoch kein Fehler generiert werden, da die Verlinkung trotzdem korrekt ist. Dasselbe gilt auch für den Tag `architect:wikidata`. Auch Architekten (oder deren Büros) dürfen eingetragen werden, die nun ersetzt worden sind. Es handelt sich dabei um eine historische Information, welches Architektur-Büro oder welcher Architekt das Bauwerk entworfen hat. Diese Aussage hat nicht den Anspruch, dass das Büro oder der Architekt noch in derselben Form existieren muss.

Aktueller Stand

Der Check hat sich als sehr komplex herausgestellt und benötigt viele Ausnahmen und Anpassungen an Spezialfälle. So sind wegen der erwähnten Zweideutigkeit von Wikidata-Artikeln (Verwaltungsgebiet und Dorf in einem) alle OSM-Objekte mit den Tags `boundary=administrative` und `place=village` generell vom Check ausgenommen.

Der Check erreicht aktuell eine True Positive-Rate von etwa 70% und ist deshalb zurzeit deaktiviert. Er erfüllt die geforderte Genauigkeit nicht und generiert momentan zu viele False Positive.

3.3.11 Dissolved, abolished or demolished

Der Check prüft, ob im verknüpften Wikidata Item das Feld *dissolved, abolished or demolished date* (P576) vorhanden ist.

Idee

Mit diesem Auflösedatum wird der Zeitpunkt angegeben, wann etwas aufgelöst oder zerstört wurde. Die Grundidee des Checks ist, dass auf OSM keine veralteten Daten abgebildet werden sollten. Zum Beispiel zeigt die Relation *Süntelbahn* (Relation 2929057) auf eine Bahnstrecke, welche gemäss Wikipedia[23] im Jahr 1995 endgültig stillgelegt wurde. Korrekterweise müsste sie deshalb

mit den Tags `railway=disued` oder `disused:railway=rail` markiert werden, was jedoch nicht der Fall ist.

Probleme

Oft sind Burgruinen wie die *Burg Ramschwag* (Q1013570) korrekterweise mit diesem Attribut versehen. Auch Objekte, welche in der Vergangenheit einmal zerstört wurden, jedoch wieder aufgebaut sind tragen oft diesen Claim. Ein solches Beispiel ist der *Bahnhof Berlin Pichelsberg* (Node 26943869) über den Wikipedia sagt:

> *Nach dem S-Bahn-Streik im Jahr 1980 wurden die Strecke und der Bahnhof stillgelegt. Mit einem neu erbauten Eingang wurde die Station am 16. Januar 1998 wiedereröffnet. (Wikipedia, 2022a)*

Aktueller Stand

Es gibt unzählige weiter solcher Beispiele, sodass der Check aktuell deaktiviert ist, um nicht zu viele False Positives zu generieren. Er müsste auf sehr spezifische Objekte oder Regeln angepasst werden, wo dieses Attribut tatsächlich nicht erlaubt ist.

3.4 Datenbankmodell

Dieses Kapitel zeigt die Abbildung 3.7 die entworfenen OR-Modell für OSM, Wikidata sowie Redirects. Relevante Aspekte werden im Anschluss beschrieben.

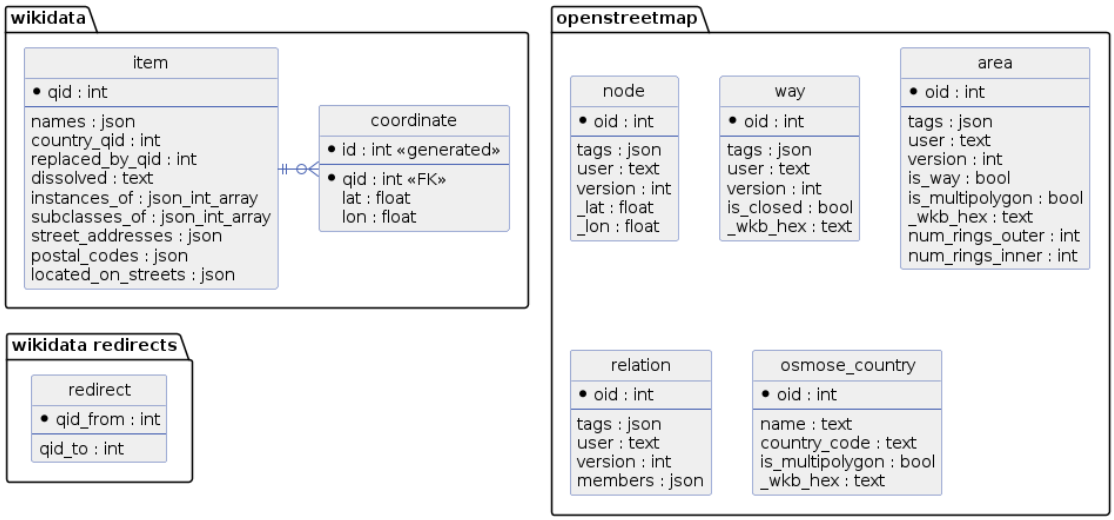


Abbildung 3.7: Datenbank Diagramm

3.4.1 OpenStreetMap

In die OSM-Datenbank werden lediglich Objekte importiert, welche mindestens ein Wikidata-Tag besitzen. Neben den relevanten objekt-spezifischen Attributen haben alle Tabellen `user`, `version` und `tags` gemeinsam. Die Pyosmium Library (siehe Unterabschnitt 4.4.1) übergibt die Tags eines Objektes als Dictionary (Key-Value-Datenstruktur). Da die Anzahl und Form der Tags variiert und

unbekannt ist, werden diese im JSON Format in die Datenbank aufgenommen. Der OR-Mapper erstellt daraus bei der Verwendung des Objektes wieder ein Dictionary.

Neben den OSM-nativen Objekten Node, Way und Relation (siehe Unterabschnitt 2.1.2) existieren in der Datenbank auch Areas. Pyosmium erstellt diese Abstraktion für alle Ways und Relations, welche einen geschlossenen Weg bilden. Der Vorteil dieser Abstraktion ist, dass die Geometrie direkt im WKB-Format vorliegt und nicht nur wie bei Relations als Referenz auf andere OSM-Objekte. Durch die Eigenschaft `is_way` lassen sich Way-Areas von Relation-Areas unterscheiden. Somit befinden sich in der Tabelle `way` nur Wege, welche keine Fläche besitzen (z.B. Strassen, Rundlaufbahn) und in `relation` nur noch Relation mit Sammlungen solcher Wege (z.B. Strassennetze) oder andere Relations.

In der Tabelle `osmose_country` sind alle Teil-Länder vorhanden, für die Osmose einzelne Error-Listen erwartet (Unterabschnitt 4.3.2).

3.4.2 Wikidata

In die Wikidata Datenbank werden sämtliche in den Dumps enthaltenen Objekte importiert. Dabei werden jedoch nur für Checks relevante Informationen aufgenommen und so die Datenlast erheblich reduziert.

Wie in der OSM-Datenbank werden Werte mit unbekannter Anzahl im JSON-Format gespeichert. Lediglich die Koordinaten werden als separate Tabelle gespeichert und sind somit auch indexiert. Dies ist nötig, um die in Abschnitt 2.2.1 spezifizierte Anforderung für Vorschläge zu erfüllen. Aktuell ist diese Anforderung jedoch nicht umgesetzt.

Es existieren bereits Checks, welche die *instance of* (P31) und/oder die *subclass of* (P279) eines Wikidata-Item überprüfen. Dafür müssen diese auch abgespeichert werden. In der Datenbank werden beide Listen je als ein JSON-Array abgespeichert (Int-Values).

3.5 Umgang mit grossen Datenmengen

Wie man in Abbildung 3.1 erkennt, gibt es im Ablauf des Programms mehrere Aufgaben, welche parallel erledigt werden können. Zudem sorgen auch Optimierungen im Code für eine Effizienzsteigerung. Dadurch lässt sich die Durchlaufzeit der Applikation effektiv verkürzen und die gegebene Infrastruktur effizient ausnützen.

3.5.1 Download und Import

Der Download und Import der Daten-Dumps kann jeweils parallel gemacht werden (Abbildung 3.8). Es kann konfiguriert werden (Abschnitt 3.6), welche Downloads oder Imports gemacht werden sollen, da Daten eventuell schon vorhanden sind.

Die beiden komplexeren Imports sind folgend jeweils noch genauer beschrieben.

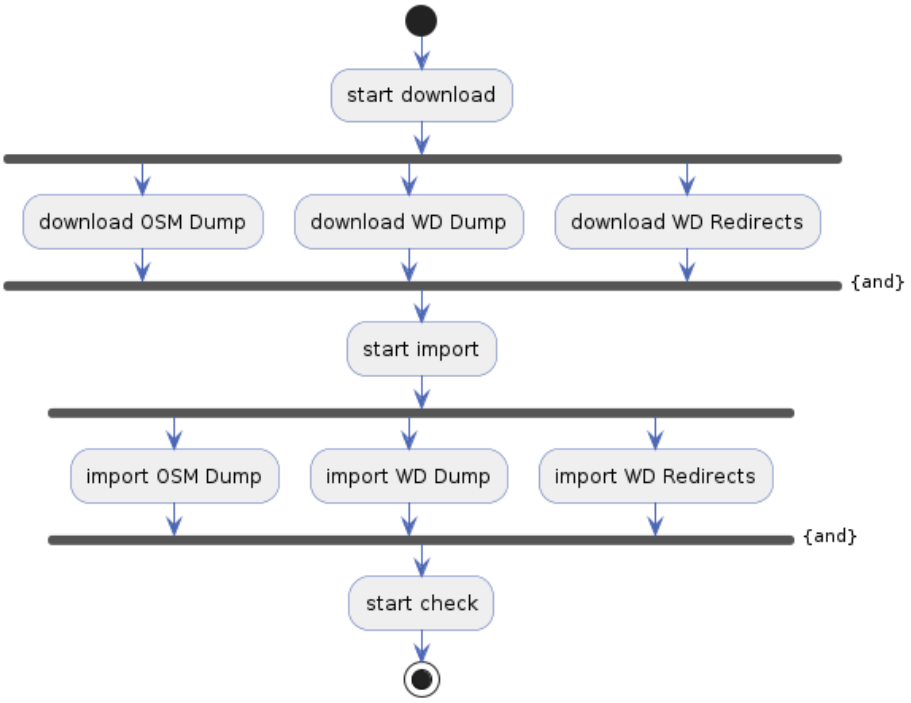


Abbildung 3.8: Download und Import Prozesse

OpenStreetMap Import

Beim Import von OpenStreetMap werden sehr viele Datensätze aus dem PBF-File geladen. Wirklich importiert werden jedoch nur die OSM-Objekte, welche ein `*wikidata` Tag besitzen. Die im PBF-File gelagerten Daten sind komprimiert ca. 65 GB gross. Dekomprimiert erreichen sie ca. 1.5 TB. Um diese Datenmenge effizient zu verarbeiten, werden beim Import drei Prozesse für das Lesen der PBF-Datei erstellt. Pyosmium ist in der Lage, nur ausgewählte Klassen zu importieren. So importiert je ein Prozess folgende OSM-Klassen:

- Nodes
- Relations
- Ways und Areas

Die Ways und Areas werden zusammengefasst, da für den Import dieser Objekte in Pyosmium ein Index mit allen betroffenen Datenpunkten gemacht werden muss. Durch die Zusammenfassung von Ways und Areas muss dieser Index nur einmal berechnet werden.

In Abbildung 3.9 kann man erkennen, dass die Import-Prozesse, die importierten Daten wiederum in eine Queue stellen, welche vom Datenbank-Füller-Prozess konsumiert wird. Alle Import-Prozesse greifen auf dasselbe PBF-File zu, da nur Read-Only darauf agiert wird, stellt das kein Problem dar.

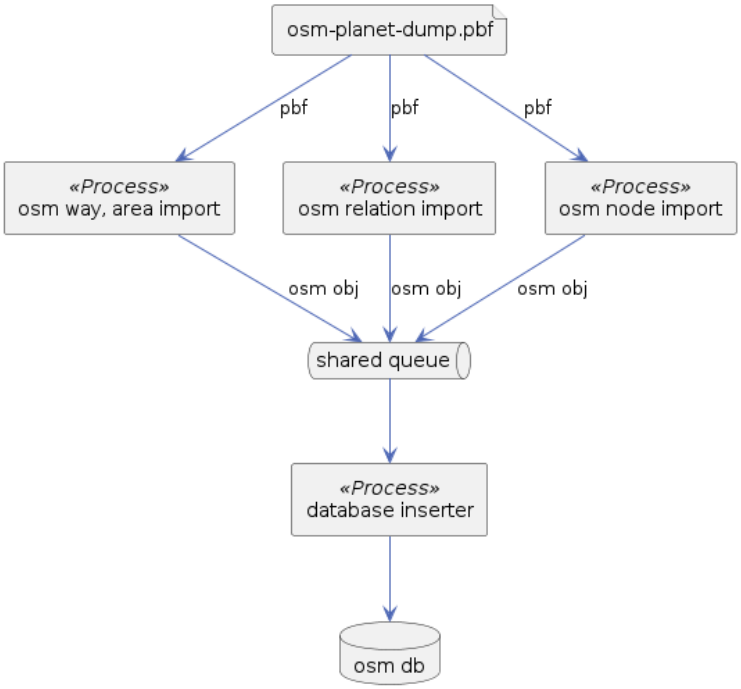


Abbildung 3.9: OSM Import Prozesse

Wikidata Import

Der Import von Wikidata ist in verschiedene Prozesse unterteilt. Wie in Abbildung 3.10 ersichtlich, ist das Einlesen und Dekomprimieren der Dump-Datei vom Einfügen in die Datenbank getrennt.

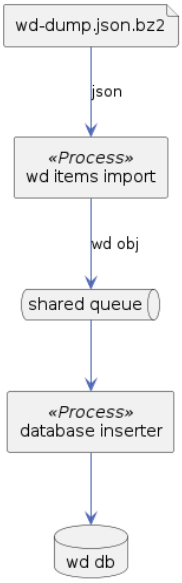


Abbildung 3.10: WD Import Prozesse

3.5.2 Checker

Da jedes OSM-Objekt für sich alleine geprüft werden kann, ist es möglich, eine beliebige Anzahl von Worker-Prozessen zu haben, welche die Checks als Jobs ausführen. Wie in Abbildung 3.11

ersichtlich, ist ein Prozess dafür zuständig, die OSM-Objekt aus der Datenbank zu laden und mehrere zusammen in Paketen in eine Queue stellt. Diese Pakete werden von den Check-Prozessen konsumiert und die darin enthaltenen OSM-Objekte verarbeitet. Die daraus resultierenden Issues werden wiederum in eine Queue gestellt, welche vom einem Reporter-Prozess konsumiert wird. Abbildung 3.11 zeigt den Zusammenhang der Prozesse grafisch auf.

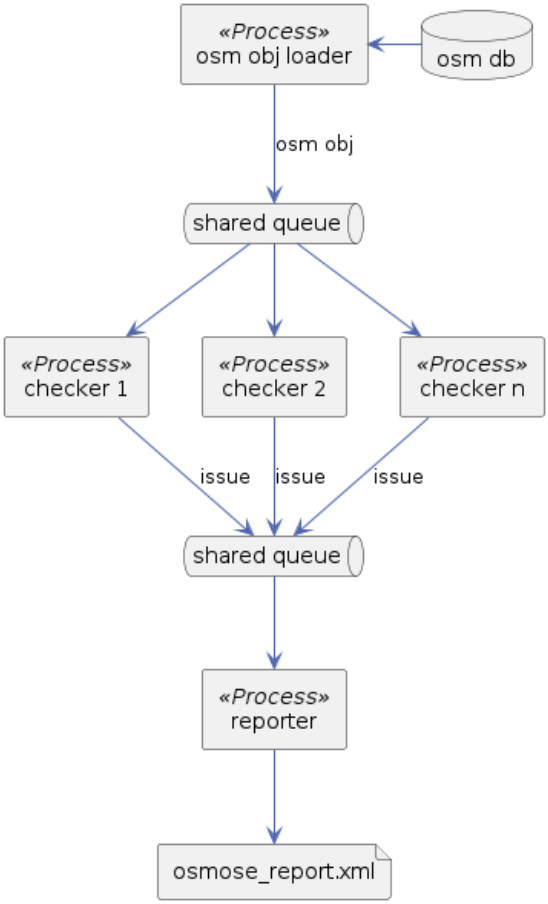


Abbildung 3.11: Checker Prozesse

3.5.3 Codeoptimierungen

In Codebereichen, welche für jedes einzelne OSM oder WD-Objekt ausgeführt werden, wird auf grösstmögliche Effizienz geachtet. Obwohl diese Microoptimierungen nur geringfügige Laufzeitverbesserung für einzelne Objekte bewirken, so ergeben sie kumuliert über die Millionen prozessierten Objekte eine signifikante Zeiteinsparung.

Beispielsweise werden die OSM-Tags als Hashtabelle (Dictionary) gespeichert, ein Key kann so in $\mathcal{O}(1)$ abgefragt werden. Wenn immer möglich, ist auf Schleifen mit einer Laufzeit von $\mathcal{O}(n)$ zu verzichten. Auch wird die relative teure Deserialisierung und Instanziierung der WKB-Rohdaten zu Geometry-Objekten erst gemacht, falls ein Check diese tatsächlich benötigt. Die erstellte Geometrie wird danach im Objekt gespeichert. Durch dieses Lazy-Loading und Caching ergeben sich ebenfalls Laufzeitvorteile.

3.6 Konfiguration und Deployment

Die Applikation wird, wie im Abbildung 3.12 ersichtlich, in einem Docker Container gestartet. Es können dabei die nachfolgend beschriebenen Konfigurationen über ein Docker-Compose-File, wie das Beispiel in Listing 3.2 zeigt, angegeben werden.

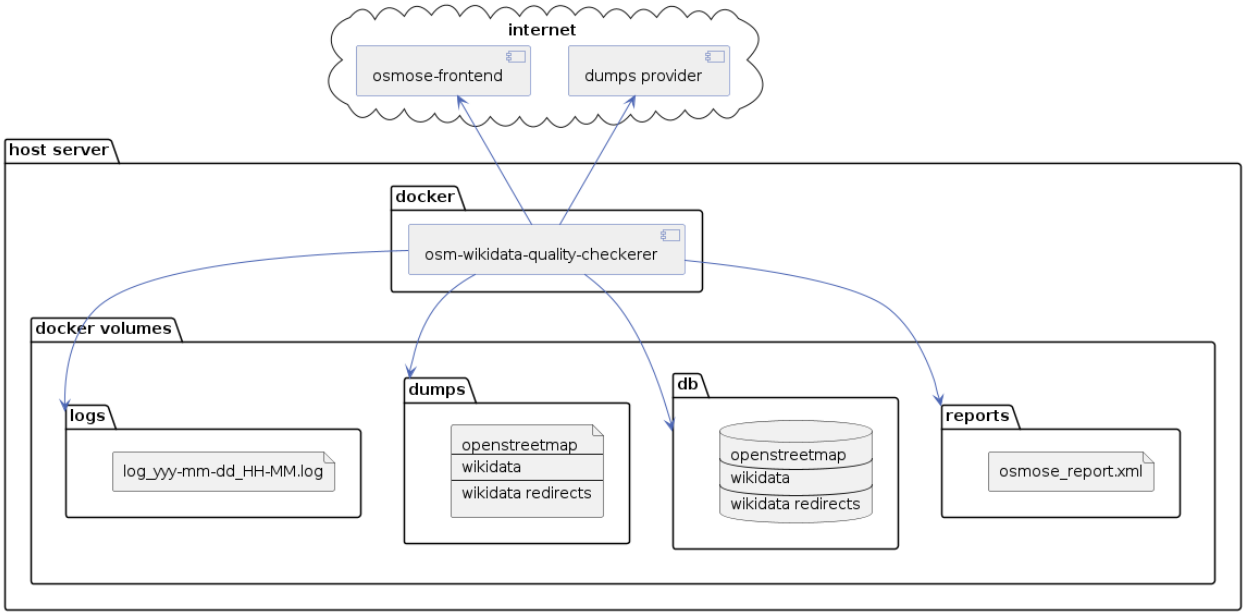


Abbildung 3.12: Deployment Diagramm

```yaml
version: "3"

services:
  osm-wikidata-quality-checker:
    image: registry.gitlab.com/geometalab/osm-wikidata-quality-checker:latest
    restart: "no"
    environment:
      OSMOSE_SOURCE: "username"
      OSMOSE_CODE: "password"
    volumes:
      - /outside_docker/dumps:/data/dumps
      - /outside_docker/db:/data/db
      - /outside_docker/reports:/data/reports
      - /outside_docker/logs:/data/logs
```

Listing 3.2: Beispiel einer Docker-Compose Konfiguration

Da das Tool in einem Docker-Container gestartet wird, existiert kein Command Line Interface (CLI). Stattdessen kann die Konfiguration über Environment Variablen gemacht werden. Im Betrieb müssen normalerweise nur die Osmose-Zugangsdaten gesetzt werden. Weitere Einstellungen sind nicht zwingend nötig, sie bieten jedoch die Möglichkeit das Programmverhalten individuell anzupassen. Dies ist vor allem für die Entwicklung sehr wertvoll.

Folgende Tabelle 3.13 beschreibt die wichtigsten Einstellungen, eine komplette Liste ist in Anhang B zu finden.

Environment Variable	Beschreibung
DO_DOWNLOAD	Spezifiziert, ob die Dumps heruntergeladen werden sollen. Default-Value ist osm,wd,redirects alle Downloads verhindern kann man mit no.
DO_IMPORT	Spezifiziert, ob die Dumps importiert werden sollen. Default-Value ist osm,wd,redirects alle Imports verhindern kann man mit no.
DO_CHECKS	Gibt an, ob die Checks gemacht werden sollen. Default-Value ist yes, die Checks abschalten kann man mit no.
OSMOSE_API_HOST	Gibt an wohin der XML-Report gesendet werden soll. Wird keine URL eingegeben, ist die Sendefunktion deaktiviert. Default-Value ist http://osmose.openstreetmap.fr/control/send-update
OSMOSE_SOURCE	Benutzername für das Osmose-Frontend
OSMOSE_CODE	Passwort für das Osmose-Frontend

Tabelle 3.13: Wichtigste Environment Variablen für die Konfiguration

KAPITEL 4

Implementation

Die folgenden Abschnitte geben Einblicke in relevante Aspekte der Implementation. Diese sind für den Unterhalt und die Weiterentwicklung relevant. Der letzte Abschnitt geht auf Softwaretests für die Sicherung der Qualität ein.

4.1 Importer

Folgend sind zwei interessante Aspekte der OSM- und WD-Import Implementation beschrieben.

4.1.1 OSM Area's

Nach Hoffmann (2022) ist eine Area ein spezielles Meta-Object, welches ein Polygon repräsentiert. Es wird von geschlossenen Ways oder Relationen (Multipolygone) abgeleitet. Areas besitzen zudem einen eigenen eindeutigen ID-Space. So wird die ID des Source-Objekts (Way, Relation) dupliziert. Im Falle von Relationen wird noch eine 1 hinzugefügt. Ob eine Area als Source eine Relation hat, erkennt man daran, dass sie eine ungerade ID besitzt. Zusätzlich existiert für denselben Zweck das boolesche Attribut `is_way`.

Wichtig zu wissen ist, das Pyosmium alle erkannten Areas zusätzlich als Way oder Relation zur Verfügung stellt. Diese sollten jedoch nur in einer Tabelle der Datenbank vorkommen, da ansonsten die Checks doppelt gemacht würden was duplizierten Fehler zur Folge hätte. Deswegen werden nur Ways in die `way`-Tabelle importiert, falls der Weg nicht geschlossen ist oder bei geschlossenen Wegen der Tag `area=no` vorhanden ist (z.B. eine Rennstrecke ist ein geschlossener Weg, jedoch keine Area). Relations werden nur in die `relation`-Tabelle aufgenommen, falls sie keinen Tag `type=boundary` oder `type=multipolygon` besitzt (z.B. Sammlung von Radwegen).

4.1.2 Aufbau der Wikidata Klassenhierarchie

Für manche Checks reichen die direkten Werte von *instance of* (P31) oder *subclass of* (P279) nicht aus (siehe Unterabschnitt 3.3.6). Sie benötigen die gesamte Klassen-/Instanz-Hierarchie. Es wird also ein weiteres `Set` benötigt, das jegliche Ursprungsklassen und -Instanzen von einem Item beinhaltet. So kann überprüft werden, ob der Wert eine Subklasse einer erlaubten Kategorie ist. Dieses Problem ist mittels rekursiver Programmierung gelöst. Listing 4.1 zeigt den entsprechenden Programmcode. Die Checks behandeln *instance of* (P31) und *subclass of* (P279) gleichwertig. Deshalb können diese beiden Felder gleich behandelt werden. Die erstelle Liste wird im Attribut `WdItem.all_instances_and_subclasses` abgelegt.

```python
def _get_recursive_instance_of(
        self, session: Session, to_load: Set[int], loaded: Set[int]
    ) -> Set[int]:

        loaded = loaded | to_load
        sub_to_load: Set[int] = set()

        rows = session.query(WdItem.instances_of, WdItem.subclasses_of).where(
            WdItem.qid.in_(to_load)
        )

        for row in rows:
            sub_to_load = sub_to_load | ((row[0] | row[1]) - loaded)

        if len(sub_to_load) > 0:
            loaded = loaded | self._get_recursive_instance_of(
                session, sub_to_load, loaded
            )

        return loaded
```

Listing 4.1: Laden der gesamten Klassen- und Instanz-Hierarchie

4.2 Checker

Die Checks können grundsätzlich in zwei Kategorien aufgeteilt werden. Die Basic Checks im Modul `checker.py` prüfen, ob die Wikidata-Tags eines OSM-Objektes mit einem validen Wert gefüllt sind und ob die entsprechenden Wikidata Items existieren und nicht weitergeleitet sind. Erst wenn diese Vorbedingung erfüllt ist, können die WD-Items aus der Datenbank geladen werden. Danach wird für jede Verknüpfung die Link Checks ausgeführt.

4.2.1 Das Checker-Modul und Basic Checks

Das Modul `checker.py` stellt die Hauptfunktion zur Verfügung, welche ein OSM Objekt überprüft und gefundene Issues zurück gibt. In Listing 4.2 ist die Signatur der Funktion zu sehen. Um dabei die nötigen WD-Items zu laden, erhält der Checker Zugriff auf die WD-Datenbank.

```python
def check(
    osm: OsmObject, wd_db: WdDatabase, wd_redirects_db: WdRedirectsDatabase
) -> CheckerIssue | None:
    # check logic
```

Listing 4.2: Signatur der Check-Funktion

Weiter macht die Funktion alle Basic Checks und führt anschliessend für jeden Link zwischen OSM und Wikidata die Link Checks aus. Die Basic Checks sind die in Kapitel 3 beschrieben Checks *Invalid Format*, *Not Existing Item* sowie *Redirected*.

4.2.2 Link Checks

In der Datei `link_checks.py` sind jene Checks enthalten, welche für jeden Link zwischen OSM und Wikidata ausgeführt werden. Ein Link ist als einzelne Verbindung zwischen OSM und Wikidata definiert. Hat ein OSM-Knoten zum Beispiel die Tags `wikidata=Q1;Q2` sowie `brand:wikidata=Q3`, so sind das drei einzelne Links, welche separat geprüft werden.

Ein Link ist als abstrakte Basisklasse (ABC) mit dem in Listing 4.3 zu sehende Interface definiert. Für jede OSM Art gibt es zur Unterscheidung eine Ableitung, wie zum Beispiel `NodeLink` für `OsmNodes`.

```python
@dataclass
class Link(ABC):
    osm: OsmObject
    wd: WdItem
    key: str

@dataclass
class NodeLink(Link):
    osm: OsmNode
```

Listing 4.3: Interface der Link-Klassen

Jeder Link Check erwartet das abstrakte Link-Objekt als einziges Argument. Der Check gibt entweder eine Issue-Instanz oder, falls kein Fehler gefunden wurde, `None` zurück. Falls ein Check nur für bestimmte OSM Objekte und/oder Tags ausgeführt werden soll, dann soll dies mit Guards am Anfang der Funktion gelöst werden. Listing 4.4 zeigt ein Beispiel einer Link Check Funktion.

```python
def check_example(link: Link) -> ExampleIssue | None:
    # do this check only for OSM nodes
    if not isinstance(link.osm, OsmNode):
      return None

    # do this check only for primary brand:wikidata
    if not osm.key == "brank:wikidata"
      return None

    # to a particular check ...
    if ...:
      return ExampleIssue(...)

    # no issue found
    return None
```

Listing 4.4: Beispiel einer Link Check Funktion

Neu hinzukommende Checks sollten diesem Muster folgen. Eine Reihe von sinnvollen und von mehreren Checks nutzbaren Funktionen sind in den Dateien `checks_common.py` ausgelagert.

4.3 Osmose Reporter

Beim Erstellen des XML für das Osmose-Frontend, müssen die Vorgaben von Osmose eingehalten werden, dabei gibt es einige Aspekte speziell zu beachten.

4.3.1 Osmose Subclass ID

Jedes Issue muss von Osmose eindeutig identifiziert werden können. Osmose löscht bei einem Update alle gemeldeten Fehler aus ihrer Datenbank und übernimmt die neue Liste. Dabei sollen aber die von der Community als False Positive markierten Fehler nicht erneut erscheinen. Osmose stellt dies über die pro Issue eindeutige Subclass-ID sicher (eindeutig in Kombination mit Class-ID). Die Subclass-ID kann vom Analyser, also der Fehler generierenden Applikation, selbst definiert werden. In diesem Projekt wird ein Hash mit dem betroffenen Key (z.B. `brand:wikidata`) und der Q-Id erstellt. Dadurch erhält ein Issues immer dieselbe Subclass-ID und wird im Falle eines False Positive nicht erneut angezeigt, bis sich an der Verknüpfung etwas geändert hat und eine erneute Prüfung erfolgen sollte. Eine vereinfachte Version dieser Hash-Generierung ist in Listing 4.5 zu sehen.

```
1  class CheckerIssue(ABC):
2      def __init__(slef, key: str, qid: int /*...*/):
3          // ....
4          self._create_hash(f"{self.key}:{self.qid}")
5
6      def _create_hash(self, to_hash: str):
7          self.hash_id = hash(to_hash) % (2**32)
8      // ....
9
10 class InvalidWdItemFormatIssue(CheckerIssue):
11     def __init__(self, key: str, invalid_qid: str /*...*/):
12         // ....
13         self._create_hash(f"{self.key}:{invalid_qid}")
14     // ....
```

Listing 4.5: Vereinfachte Generierung des Hashes für die Osmose Subclass ID

4.3.2 Reports nach Regionen aufgeteilt

Osmose wünscht die Lieferung der Fehler nach definierten Ländern oder Regionen (z.B. in der Schweiz Kantone) aufgeteilt. Diese Aufteilung geschieht zu Statistikzwecken und um den lokalen Communities einen besseren Überblick zu bieten. Die Liste mit den Regionen und deren OSM-Relation-ID kann aus einem OsmoseConfig-File[24] ausgelesen werden. Die benötigten Relationen werden beim OSM-Import in die Tabelle `osmose_country` geladen. Die Liste aller Länder wird nur vom Osmose-Reporter benötigt. Dieser sucht damit für jeden gefundenen Fehler das passende Land, indem er prüft, ob die Koordinate sich im oder auf dem Polgyon befindet.

Es kann vorkommen, dass für ein Fehler keine Region gefunden wird und dieser deshalb nicht an Osmose gemeldet werden kann. Dies ist oft darauf zurückzuführen, dass ein Polygon der Region nicht geschlossen oder anderweitig defekt ist und das Objekt deshalb nicht als Area importiert werden kann. Solche Fehler werden von anderen Osmose-Analyser überprüft und sollten zeitnah korrigiert werden. Ist die Integrität des Objektes wieder hergestellt, funktioniert beim nächsten Durchlauf der Applikation auch das Reporting für die betroffene Region wieder.

4.3.3 Koordinaten für Relations

Osmose benötigt für jeden Fehler eine Location, um diesen auf der Karte des Frontends platzieren zu können. Bei OSM-Nodes, -Ways und -Areas liegt dies entweder direkt als Koordinate oder via dem Mittelpunkt der Geometrie vor. Bei reinen Relations, welche nur eine Sammlung von anderen Objekten darstellen, liefert Pyosmium jedoch keine geografischen Angaben. Aktuell werden deshalb Fehler zu Relations, welche keine Area darstellen, nicht an Osmose gesendet. Dieses Thema wird als Vorschlag für die Weiterentwicklung in Unterabschnitt 5.2.3 behandelt.

4.4 Externe Libraries

Folgend werden die wichtigsten verwendeten externen Libraries beschrieben.

4.4.1 Pyosmium

Pyosmium[25] ist ein Python-Modul, das einen Zugang zu einigen Funktionen der Osmium Library[26] (C++) ermöglicht. Mit dieser können OSM Entites (Nodes, Ways, Relations, Areas und Changesets) aus OSM PBF Files eingelesen werden. Dieses Modul wird beim Import des OSM Dumps verwendet. Geometrische Objekte werden dabei von Pyosmium in das WKB Format übersetzt, um sie so in der Datenbank zu speichern.

4.4.2 Shapely

Shapely[27] kann geometrische Objekte analysieren und manipulieren. Es ist in der Lage, das von Pyosmium gelieferte WKB-Format zu verarbeiten. Diese Library wird von den Checks verwendet, zum Beispiel lässt sich so herausfinden, ob eine Koordinate innerhalb eines Polygons liegt oder nicht.

4.4.3 Wikidata Client Library

Wikidata Client Library[28] lädt Wikidata Items direkt von der offiziellen API. Sie wird nur von den Basic-Checks (Unterabschnitt 3.3.2 und 3.3.3) eingesetzt, wenn ein Item weder in der vom Dump generierten Datenbank noch in der Redirect-Liste gefunden werden kann. Durch diese Abfrage kann sichergestellt werden, dass dieses Verhalten nicht auf einen zu alten Dump zurückzuführen ist.

4.4.4 SQLAlchemy

SQLAlchemy[29] ist ein SQL Toolkit für Python. Es unterstützt verschiedene Datenbanksysteme, darunter SQLite. Es bietet zudem eine Object Relational Mapping (ORM) Funktionalität an, welche von diesem Tool verwendet wird.

4.4.5 FuzzyWuzzy

FuzzyWuzzy[30] ist eine Python-Bibliothek mit vielen hilfreichen Funktionen für String-Vergleiche. Es verwendet den Levenshtein-Distanz Algorithmus zur Berechnung der Unterschiede zweier Strings mit einem einfach zu bedienenden Interface.

4.5 Unit-Tests und Testabdeckung

Beim Testing wird primär auf automatisierte Tests gesetzt. Diese können bei Code-Änderungen sehr schnell einen Überblick geben, ob die Software noch so funktioniert wie erwartet. Die automatisierten Tests können lokal ausgeführt werden und sind in der Continuous Integration (CI)-Pipeline (siehe Unterabschnitt 7.2.2) eingebaut.

Für die Unit-Tests wird pytest[31] verwendet. Pro Modul existiert ein Test-File mit den Unit-Tests. Die Unit-Test werden mit den Konsolenbefehl `poetry run pytest` ausgeführt.

Die Testabdeckung wird mit dem Tool coverage[32] berechnet. Dieses ist auch in der CI-Pipline integriert und liefert so bei einem Merge Request direkt die aktuelle Testabdeckung. Ein Testabdeckungsrapport ist im Anhang E zu finden.

KAPITEL 5

Resultate

Dieses Kapitel zeigt die erreichten Ziele und unter welchen Bedingungen das Tool eingesetzt werden kann. Abschliessend werden Vorschläge für mögliche Weiterentwicklungen des Programms gemacht.

5.1 Zielerreichung

Dieser Abschnitt beinhaltet die Bewertung der Ziele, welche in den funktionalen und nicht-funktionalen Anforderungen definiert sind.

5.1.1 Gefundene Fehler und Accuracy

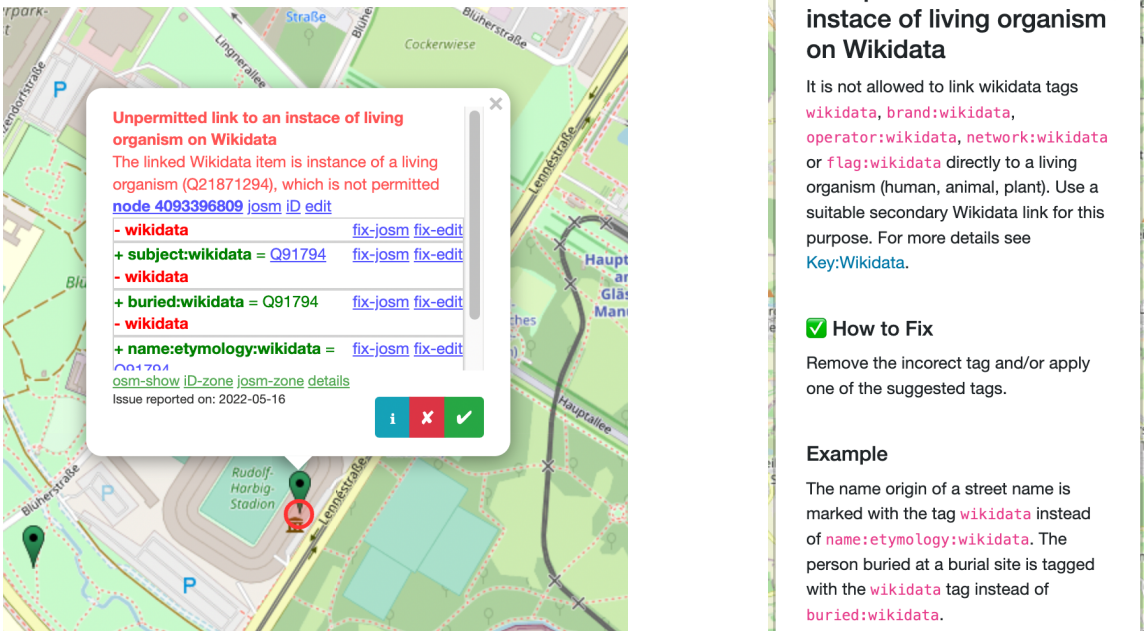
In den aktivierten Checks werden über 30'000 Fehler gefunden. Die Checks sind dabei sehr präzise ausgelegt und erreichen eine True-Positive-Rate von 95%. Diese hohe Treffsicherheit ist nötig, um Osmose nicht mit tausenden False Positives zu fluten. Es wird dabei jedoch in Kauf genommen, dass es False Negatives gibt, welche zurzeit unerkannt bleiben.

Fehlerkategorie	Node	Way	W-Area	Relation	R-Area	Total	Accuracy
Invalid Format	248	144	139	11	13	555	100%
Not Existing Item	601	645	323	88	106	1763	99%
Redirected Item	6355	817	3192	307	1543	12214	100%
Unpermitted Instance	1313	995	920	168	60	3456	95%
Living Organsim	5731	649	514	267	83	7244	95%
Primary Tag Claim	197	0	196	4	36	433	90%
Secondary Tag Claim	1895	1071	1344	119	4	4433	90%
Place Mismatch	398	0	41	0	32	471	95%
Very Large Distance	204	0	95	0	19	318	95%
Total	**16942**	**4321**	**6764**	**964**	**1896**	**30887**	**95%**

Tabelle 5.1: Statistik über die gefundenen Fehler

5.1.2 Meldung an Fehlerdatenbank und Korrekturvorschläge

Die gefunden Fehler werden am Ende des Prozesses automatisch im XML-Format an das Osmose-Frontend gesendet und dort in die Osmose Fehlerdatenbank eingespeist. Dabei werden je nach Fehlerkategorie sinnvolle Korrekturvorschläge gemacht. Ein Beispiel ist in Abbildung 5.1a zu sehen, wo ein Gedenkstein (Node 4093396809) direkt auf die betreffende Person (Q91794) zeigt. Mit dem ersten Korrekturvorschlag auf `subjekt:wikidata` kann das Problem mit einem Klick behoben werden. Ausserdem werden dem Benutzer bei allen Fehlerkategorien die in Abbildung 5.1b sichtbaren Zusatzinfos wie Fehlerbeschreibung, Korrekturhinweise und Beispiele angezeigt.



(a) Anzeige und Korrekturvorschläge (b) Zusatzinformationen

Abbildung 5.1: Anzeige der Fehler im Osmose-Frontend

Wie in Unterabschnitt 4.3.3 beschrieben, besteht gegenwärtig eine Einschränkung bei Fehler, welche sich auf Relations ohne Area beziehen. Für diese können in den meisten Fällen keine Koordinaten berechnet werden. Da die Angabe der Koordinaten für das Osmose-Frontend zwingend ist, um den Fehler auf der Karte darzustellen, können reine Relationen nicht der Osmose-Fehlerdatenbank gemeldet werden. Das betrifft rund 900 Objekte oder 3% aller Fehler. Vorschläge, wie dieses Problem in Zukunft behoben werden könnte, werden in Unterabschnitt 5.2.3 aufgezeigt.

5.1.3 Monitoring und Logging

Das Feedback zum aktuellen Stand der Applikation wird über die Kommandozeile ausgegeben und kann über Docker angezeigt werden. Zudem werden alle Ausgaben in einem Logfile ausserhalb des Dockers protokolliert. So werden diese Informationen gesichert und können benützt werden, um Fehler oder Veränderungen im Ablauf zu erkennen und zu beheben.

5.1.4 Performance

Ein kompletter Durchlauf dauert auf dem zur Verfügung gestellten Server rund 24h Stunden und teilt sich in die in Tabelle 5.2 sichtbaren Zeiten auf. Bei diesen handelt es sich um Durchschnittswerte mehrerer Runs. Zum Vergleich wurden Messungen auf einem schwächeren 8-Core System gemacht, sowie ohne Multiprocessing bei den Checks. Da auf den Systemen noch andere Anwendungen liefen, sind diese Zahlen lediglich als Richtwerte zu verstehen.

Das Ziel wurde übertroffen und die gesamte Welt kann problemlos innerhalb einer Wochenfrist geprüft werden. Es ist genügend Laufzeitreserve verfügbar, sodass dieses Ziel auch auf schwächeren Infrastrukturen eingehalten werden kann. Die gute Performance wird vor allem durch Ausnutzung von Multiprocessing (siehe Abschnitt 3.5) erreicht. Beim Checker-Teil wird so ein markanter Speedup von 20x erreicht. Die kurze Check-Zeit ist vor allem für die Entwicklung von

Checks hilfreich. So können schneller mehrere Iterationen von Prüfung und Codeanpassungen durchgeführt werden.

Schritt	**32-Core**	**8-Core**	**1-Core**
Download	5h	5h	-
Import	18h	24.5h	-
Check	1h	10.5h	20h
Total	**24h**	**40h**	-

Tabelle 5.2: Durchschnittliche Laufzeiten

Für die Dumps sowie Datenbanken muss der in Tabelle 5.3 sichtbare Speicherplatz zur Verfügung gestellt werden. Diese Datenmengen stellen für normale Infrastrukturen kein Problem dar und sind mehrere Grössenordnungen kleiner als der Betrieb und Unterhalt einer vollständigen Kopie beider Datenbanken. Durch die ständige Erweiterung beider Datenbanken sowie der wachsenden Popularität der Wikidata-Tags muss jedoch mit einem Datenwachstum gerechnet werden, daher sollte genügend Reserve bereitgestellt werden.

Artefakt	**Grösse**
OSM Dump (planet)	64 GB
WD Dump	70 GB
OSM DB	16 GB
WD DB	38 GB
Diverses (Redirects, Reports)	$<$1 GB
Total	**189 GB**

Tabelle 5.3: Speicherbedarf

5.1.5 Reliability (Fault Tolerance)

Das Tool ist so konzipiert, dass es bei geringfügigen Fehlern weiterläuft und wird eine hohe Fehlertoleranz erreicht. Laufzeitfehler beim Importieren der Dumps oder während dem Ausführen von Checks, werden von der Applikation in den Logs protokolliert. Sie führen jedoch nicht zum kompletten Abbruch des Programms. So können alle gültigen Objekte dennoch geprüft werden.

Zum Beispiel kann es bei Areas vorkommen, dass die Linien des Polygons nicht geschlossen sind oder sich überkreuzen. Die Abbildung 5.2 zeigt ein Beispiel eines defekten Polygons. Da andere Osmose-Analysers[33] gezielt nach solchen Problemen suchen, werden diese meist rasch repariert, sodass diese in einem späteren Durchlauf der Applikation korrekt importiert und geprüft werden können.

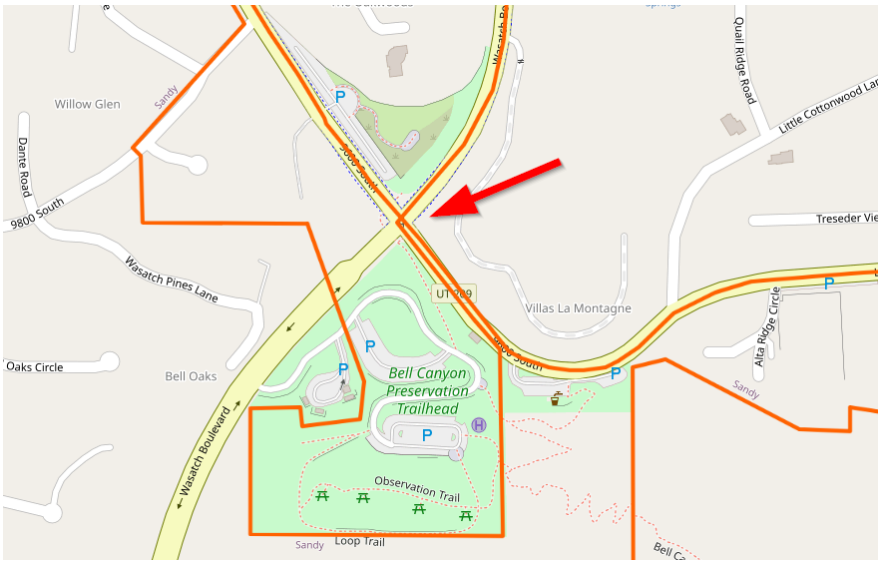


Abbildung 5.2: Defektes Polygon wegen überkreuzender Linien in der Relation 199010 (vers. #13)

5.1.6 Maintainability, Operabillity und Testabdeckung

Die überlegte Architektur mit klaren Verantwortlichkeiten und wenig Abhängigkeiten hilft, die Software weiter auszubauen. Die Dokumentation (README), welche in Anhang B zu sehen ist, ermöglicht es neuen Entwicklern, sich schnell im Code zurechtzufinden. Wie in Unterabschnitt 4.2.2 beschrieben, sind die Checks zudem so abstrahiert, dass diese auf einzelnen Verknüpfungen von OSM zu Wikidata funktionieren, was das Design der Checks erheblich einfacher macht, als die Verwendung roher OSM-Objekte.

Eine hohe Testabdeckung von über 80% durch Unit-Test sowie den zusätzlichen statischen Codeanalysen in der CI-Pipeline (Anhang C) helfen, die Qualität der Software zu erhalten.

5.2 Ausblick

Weiterhin gibt es viele denkbare Erweiterungen, um den Funktionsumfang und die Genauigkeit der Checks zu erhöhen. Folgende Weiterentwicklungen sind sehr interessant und im Bereich des Möglichen:

5.2.1 Checks weiterentwickeln und ausbauen

Die implementierten Checks zeigen ein breites Spektrum von Möglichkeiten. Diese lassen sich jedoch weiter verbessern und ausbauen.

Dissolved und Replaced-by Checks

Der Replaced-by (Unterabschnitt 3.3.10) und der Dissolved-Check (Unterabschnitt 3.3.11) sind aktuell deaktiviert. Sie erzeugen noch zu viele False Positive Fehlermeldungen. Mit einer umfassenden Analyse wäre es aber eventuell möglich, diese so zu verbessern, so dass diese qualitativ genügende Resultate liefern.

Primary Tag Mismatch Check

Der Primary Check deckt im Moment nur einen Bruchteil der möglichen Anwendungsfälle ab. Es ist möglich, diesen auf viele weitere Parameter auszuweiten. Jeder neue Parameter muss jedoch sorgfältig analysiert werden, um die nötige Qualität und Genauigkeit einzuhalten.

Place Mismatch Check

Weitere spezifische Checks, die dem Beispiel des Place-Checker (Unterabschnitt 3.3.8) folgen, könnten interessante und nützliche Anwendungsbereiche abdecken. Zum Beispiel könnten ähnliche Checks für POIs's wie Nationalparks oder Krankenhäuser erstellt werden.

5.2.2 Lexeme Check

Lexeme werden aktuell bei dem Tag `name:etymologie:wikidata` toleriert jedoch nicht weiter überprüft. Falls sich diese Art von Verknüpfung in OpenStreetMap durchsetzen sollte, könnten die Lexeme ebenfalls in die Wikidata-Datenbank importiert und überprüft werden.

5.2.3 Koordinaten für Relations

Bei reinen Relationen, die nicht als Area interpretiert werden, gibt Pyosmium keine Koordinaten an. Für die Lieferung von Fehler an Osmose, sind diese jedoch vorgeschrieben. Es muss also ein Weg gefunden werden, wie die Koordinaten entweder beim Import in die Datenbank gespeichert werden können oder beim Generieren eines Fehlers über eine API abgefragt werden. Aktuell werden Fehler auf solchen Relationen nicht an Osmose gesendet.

5.2.4 Vorschläge für Verknüpfungen generieren

Die Abdeckung von Verknüpfungen ist stetig am Steigen. Es ist jedoch wünschenswert, dass weitere Verknüpfungen hinzugefügt würden. Da fast alle notwendigen Daten in den beiden importierten Datenbanken vorhanden sind, wäre es sehr interessant, daraus auch Vorschläge für weitere Verknüpfungen zu generieren. Um sich auf wichtige Gebiete und Wikidata-Items zu konzentrieren, könnten die Services von Sascha Brawer für ein Ranking verwendet werden. Dies sind die osmviews-py[34], um ein Ranking für eine Location zu erhalten und Wikidata QRank[35] für ein Ranking von Wikidata-Items.

KAPITEL 6

Schlussfolgerung

Die gesetzten Ziele wurden vollständig erreicht und die Applikation leistet einen wertvollen Beitrag zur Qualität-Sicherung auf OpenStreetMap. Durch die Konzentration auf präzise Checks, ist das Tool in der Lage, Fehler in den Verknüpfungen zu Wikidata mit hoher Treffsicherheit von 95% zu erkennen und an das Osmose-Frontend zu melden. Es werden insgesamt über 30'000 Fehler in neun Kategorien gefunden.

Zwei Probleme haben sich dabei als besonders herausfordernd herausgestellt. Einerseits war dies die Datenqualität auf Wikipedia, welche mitunter sehr lückenhaft ist. Anderseits war dies die offene Architektur von OpenStreetMap, die viele unterschiedliche Möglichkeiten zur Kartierung eines Objekts zulässt, welche alle beachtet werden müssen.

Durch die eingesetzte Import-Logik und das eigene Datenbankmodell, bei welchem nur die relevanten Daten extrahiert werden, kann die Applikation mit den grossen Rohdaten umgehen und die ganze Welt prüfen. Das Tool läuft in einem Docker-Container, besorgt sich selbstständig die Datensätze und liefert Artefakte wie Reports und Logs als Output. Somit ist ein automatisierter und regelmässiger Betrieb möglich. Auf dem Referenzsystem beträgt die Laufzeit des Programms von Anfang bis Ende rund einen Tag. Auch auf schwächeren Servern kann eine Laufzeit von unter einer Woche eingehalten werden. Die ausführliche Dokumentation sowie eine offene und leicht verständliche Architektur machen es einfach, das Tool auszubauen und weitere Checks zu implementieren. Dadurch eröffnet die Applikation ein grosses Potential für Erweiterungen neuer Checks oder die Verbesserung bestehender.

Weiterhin gibt es viele denkbare Erweiterungen, um den Funktionsumfang und die Genauigkeit der Checks zu erhöhen.

KAPITEL 7

Projektmanagement

Dieses Kapitel enthält relevante Information zum Projektmanagement. Nach der Beschreibung des Entwicklungsprozesses, zeigt der Projektplan den Ablauf und die einzelnen Phasen. Risiken werden analysiert und Wege zur Mitigation aufgezeigt. Zum Schluss kommt die Zeitauswertung des Projektes.

7.1 Entwicklungsprozess

Der Entwicklungsprozess beschreibt, nach welcher Methodik die Entwicklung geplant und durchgeführt wurde. Dazu gehört der Workflow das Dokumentieren von Entscheidungen.

7.1.1 Methodik

Bei der Planung wird vom Groben ins Kleine vorgegangen. Als Grobplanung dient der Projektplan mit den Meilensteinen, der die Themen und den Zeitrahmen einer Phase vorgibt. Gearbeitet wird in wöchentlichen Iterationen. Zu Beginn jeder Iteration werden die einzelnen Tätigkeiten genauer analysiert, falls nötig in Arbeitspakete aufgeteilt und mit einer Zeitschätzung versehen. So kann kontrolliert werden, ob die Arbeitslast in den Zeitrahmen passt.

Für jedes Arbeitspaket existiert ein GitLab-Issue. Dies ermöglicht eine einfache Überwachung der aufgewendeten Zeit pro Task. Zudem kann der Entwicklungsprozess nach GitFlow organisiert werden (Unterabschnitt 7.1.3).

Um den Projektstand und Fortschritt zu messen, wird das Kanban-Board von GitLab verwendet, darin sind alle Arbeitspakete mit ihrem Status ersichtlich. Da für das Projekt ein definiertes Zeit-Budget einzuhalten ist, wird die aufgewendete Zeit gemäss der Beschreibung in Abschnitt 7.5 erfasst und periodisch ausgewertet.

7.1.2 Dokumentation von Architektur-Entscheidungen

Nach Dasanayake, Markkula, Aaramaa und Oivo (2015) sind Software-Architektur-Entscheidungen ein wichtiger Punkt in der Software-Entwicklung. Durch eine Umfrage wurde herausgefunden, dass die mangelnde Dokumentation eine der grössten Herausforderungen darstellt. Da solche Entscheide oft zu Beginn gemacht werden und zudem eine grosse Auswirkung auf die gesamte Lebensdauer der Applikation haben, ist es wichtig, diese Entscheidungen gut zu dokumentieren.

Wichtige Entscheidungen werden deshalb mit der Y-Methode nach Zimmermann (2020) erfasst. Dadurch können Entscheidungen zu einem späteren Zeitpunkt und ohne Informations-Overhead nachvollzogen werden. Die folgende Vorlage zeigt den Aufbau einer Entscheidung nach der Y-Methode:

> Y
>
> Im Zusammenhang mit *(use case or component)*, mit der Anforderung *(non-functional concern)*, entschieden wir uns für *(option 1)* und verwerfen *(option 2)*. So erreichen wir *(quality)*. Dabei wird in Kauf genommen, dass *(consequence)*.

7.1.3 Workflow

Der Workflow ist durch gute Erfahrungen mit dem GitFlow-Branching-Modell inspiriert (Pidoux, 2014).

Für jedes Feature wird ein Branch über das jeweilige GitLab-Issue erstellt. Für die Integration eines Features in den Develop-Branch muss ein Merge-Request erstellt werden. Die Merge-Requests setzen das erfolgreiche Durchlaufen der CI-Pipeline und ein Code-Review voraus. Auf dem Develop- und Release-Branch existiert somit immer eine lauffähige Version der Software. Abbildung 7.1 zeigt ein vereinfachtes Beispiel des Workflows.

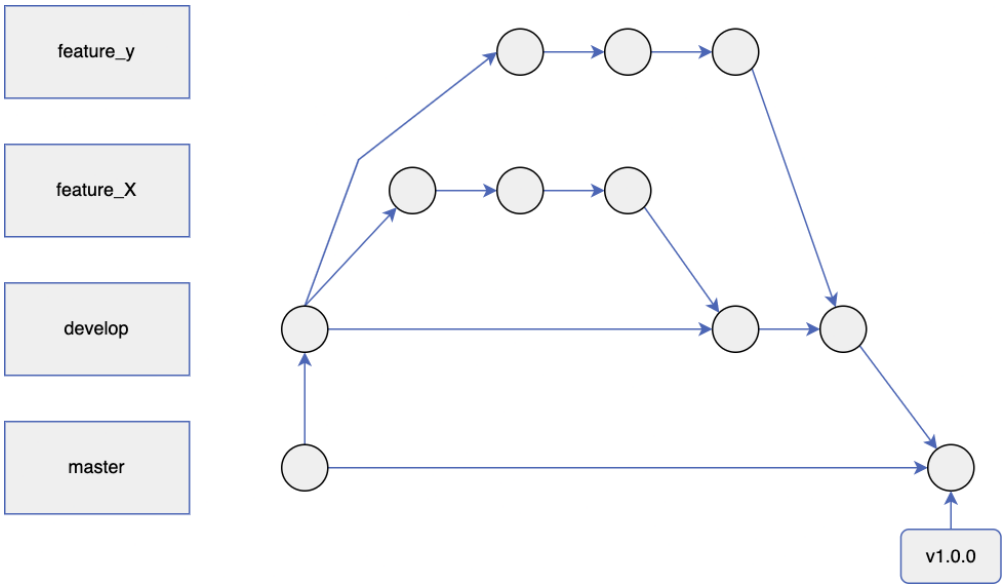


Abbildung 7.1: Git Workflow

7.2 Qualitätssicherung

Die getroffenen Massnahmen sind wichtige Instrumente zur Verbesserung und Sicherung der Qualität des Produkts. Die Kontrollen sollten nach Möglichkeit automatisiert werden. Um dies zu erreichen, wird die automatisierte CI-Pipeline von GitLab eingesetzt. Der Aufbau der Pipeline ist im Anhang C genauer beschrieben.

7.2.1 Code Reviews

Jedes Feature wird in einem eigenen Branch entwickelt. Beim Merge-Request in den Develop-Branch muss die andere Person ein Code-Review durchfuhren. Allfällige Fragen oder Bedenken müssen im Team abgesprochen werden, bevor der Merge bestätigt werden darf.

7.2.2 Automatisierte Unit-Tests

In der CI-Pipeline werden die Unit-Test automatisch ausgeführt. Dadurch können Fehler früh erkannt und korrigiert werden. Ausserdem stellt es sicher, dass auf dem Develop-Branch immer eine lauffähige Version des Programms existiert. Das Testing und die eingesetzte Library sind in Abschnitt 4.5 genauer beschrieben.

7.2.3 Testabdeckung

Es wird eine Testabdeckung der Uni-Test von 80% oder höher angestrebt.

Die Testabdeckung darf nicht als Indikator für Test-Qualität missinterpretiert werden. Gerade bei interpretierten und dynamisch typisierten Sprachen wie Python es ist jedoch sinnvoll, möglichst sämtliche Codeteile zu durchlaufen, um Typenfehler oder ungültig Funktionsaufrufe frühzeitig und automatisiert zu erkennen.

7.2.4 Code-Formatierungsrichtlinien

Als Python-Styleguide wird PEP 8[36] definiert. Das Tool Black[37] sorgt für deren Einhaltung. In der CI-Pipeline ist Black als Check eingebaut, sodass ein Merge-Request fehlschlägt, wenn die Formatierung korrekt wird.

Zusätzlich wird eine Sortierung der Imports von der Pipeline überprüft. Dafür ist das Tool isort[38] zuständig. Es sortiert und überprüft die Imports.

7.2.5 Statische Code-Analysen

Die beiden Tools pylint[39] (Linter) und mypy[40] (Static Type Check) werden für statische Code-Analysen eingesetzt, um einen möglichst stabilen Code zu gewährleisten. Beide Tools werden in der Pipeline ausgeführt. False Positives von pylint können mit entsprechender Konfiguration für das gesamte Projekt oder bestimmte Codezeilen deaktiviert werden.

7.3 Projektplanung und Meilensteine

Das Projekt ist in mehrere Phasen gegliedert. Das Ziel ist eine Grobplanung, welche in den einzelnen Phasen verfeinert wird. Die Phasenabschlüsse und die extern vorgegebenen Termine ergeben Meilensteine, welche in den folgenden Unterabschnitten erläutert werden. Tätigkeiten wie Dokumentation und Projektmanagement ziehen sich über das gesamte Projekt. Abbildung 7.2 zeigt den Projektplan als Gantt-Diagramm.

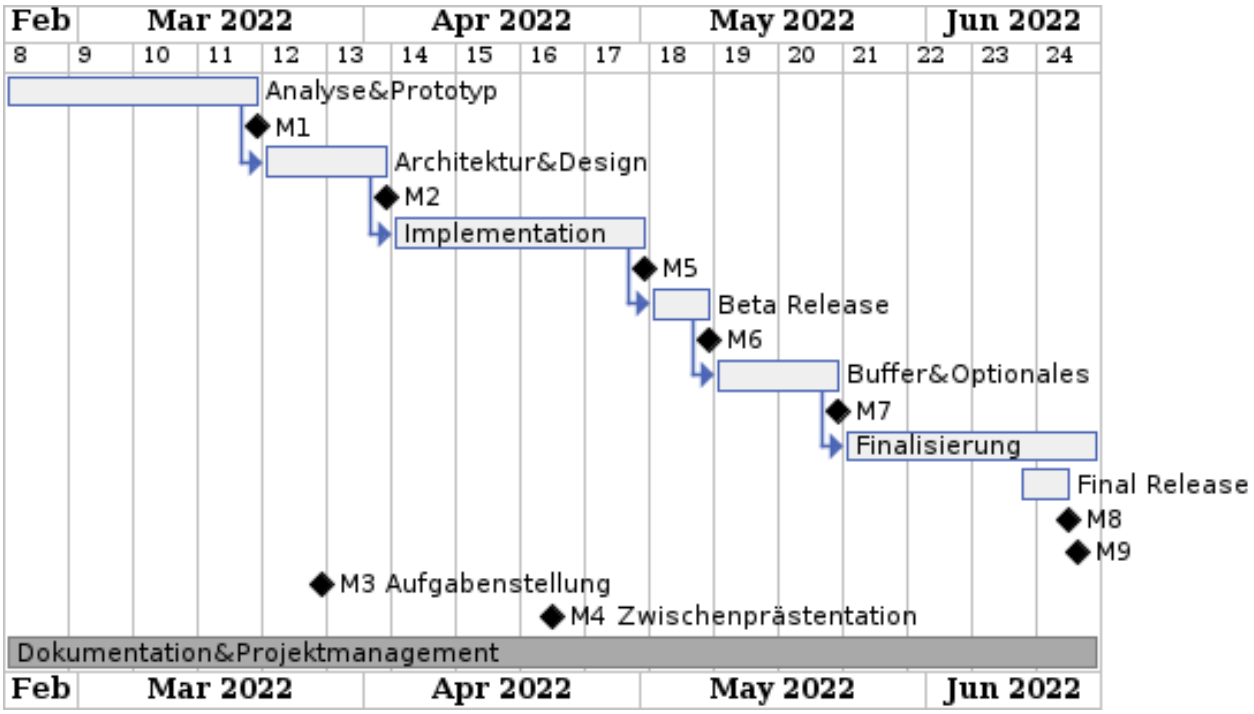


Abbildung 7.2: Gantt Projektplan

7.3.1 Analyse & Prototyp (M1)

Zeitraum: 21.02. - 20.03.2022

In der ersten Phase wird sich mit der Aufgabenstellung und der Datenstruktur von OpenStreetMap sowie Wikidata auseinandergesetzt. Verschiedene Konzepte zum Import und Indexierung der Datenbank werden ausprobiert und ein Prototyp als technologischer Durchstich erstellt. Ausserdem werden erste Checks implementiert und deren Genauigkeit manuell überprüft. Erste Reports für Osmose werden erstellen und in einer lokalen Osmose-Instanz hochgeladen.

7.3.2 Architektur & Design (M2)

Zeitraum: 21.03. - 03.04.2022

Aufgrund der gewonnen Erkenntnisse mit dem Prototyp wird die Architektur sowie das Design vor allem im Bereich der Checks definiert.

7.3.3 Aufgabenstellung (M3)

Datum 27.03.2022

Die Aufgabenstellung wird konkretisiert.

7.3.4 Zwischenpräsentation (M4)

Datum: 21.04.2022

In der Zwischenpräsentation werden dem Gegenleser und weiteren involvierten Personen, die Arbeit und der bisherige Stand sowie einen Ausblick präsentiert.

7.3.5 Implementation (M5)

Zeitraum: 04.04. - 01.05.2022

Das eigentliche Produkt wird gemäss beschlossener Architektur implementiert. Diese Phase beinhaltet vor allem folgende Punkte:

- Grundarchitektur
- Import & Datenbankmodell
- Diverse Checks
- Reporter

7.3.6 Beta Release (M6)

Zeitraum: 02.05 - 08.05.2022

Ein öffentlicher Beta Release mit funktionierenden und qualitativ guten Checks hilft dabei, die Osmose-Community von diesem Projekt zu überzeugen und erstes Feedback zu erhalten.

7.3.7 Buffer & Optionales (M7)

Zeitraum: 09.05 - 22.05.22

Hier können weitere Checks eingebaut werden oder die Genauigkeit der Bestehenden verbessert werden. Weitere optionale Features gemäss Abschnitt 2.2 können implementiert werden.

7.3.8 Final Release (M8)

Zeitraum: 10.06. - 16.06.22

Nach dem Feature Freeze wird das Produkt getestet und eine finale Version freigegeben.

7.3.9 Finalisierung (M9)

Zeitraum: 23.05. - 17.06.22

Die Dokumentation und weitere Dokumente wie Readme, Abstract und Management Summary werden fertiggestellt. Das obligatorische Poster und die Endpräsentation werden vorbereitet.

7.4 Risikoanalyse

Die Risikoanalyse identifiziert Risiken und zeigt Massnahmen zu deren Mitigation auf.

7.4.1 Risiken

Die Tabelle 7.1 listet die Risiken auf. Diese werden im Anschluss erläutert.

Nr.	Risiko	Schaden	Eintrittsw.
1	Fehlendes Knowhow im Bereich OSM und WD	Mittel	Klein
2	Skalierungsprobleme	Mittel	Mittel
3	Kein Zugang zum Osmose-Frontend	Klein	Hoch
4	Manuelle Konfiguration bei jedem Lauf nötig	Klein	Klein

Tabelle 7.1: Risiken für das Projekt

Legende Schaden:

- Hoch: Führt zum Scheitern der Arbeit
- Mittel: Verringert die Funktionalität oder den Umfang
- Klein: Wenig bis keine signifikanten Auswirkungen

Legende Eintrittswahrscheinlichkeit:

- Hoch: $> 70\%$
- Mittel: $30 - 70\%$
- Klein: $< 30\%$

#1 Fehlendes Knowhow im Bereich OSM und WD

Die Datenmodelle von OSM und/oder Wikidata sind komplizierter aufgebaut als erwartet, die Einarbeitung dauert daher länger. Es ist kein zufriedenstellendes Ergebnis in nützlicher Frist möglich.

#2 Skalierungsprobleme

Bei der Verarbeitung der grossen Datensätze gibt es Skalierungsprobleme. Diese Probleme sind nicht oder nicht in der gegebenen Zeit lösbar.

#3 Kein Zugang zum Osmose-Frontend

Osmose gewährt keinen Zugriff, um die gefundenen Fehler in ihr System einzutragen. Der praktische Nutzen des Produktes ist daher begrenzt.

#4 Manuelle Konfiguration bei jedem Lauf nötig

Das Tool lässt sich nicht wie geplant in einem Docker Container ohne Konfiguration starten und kann seine Tätigkeiten nicht automatisiert ausführen. Es ist eine manuelle Konfiguration bei jedem Lauf nötig.

7.4.2 Mitigation

#1 Fehlendes Knowhow im Bereich OSM und WD

Gute Vorbereitung mittels der von OSM und WD zur Verfügung gestellten Dokumentation und Tutorials. Frühen MVP Prototyp entwickeln, um so einen Proof of Concept zu erhalten.

#2 Skalierungsprobleme

Schon im Prototyp sollen Skalierungstests mit den kompletten Datensätzen (ganze Welt) gemacht werden. So können Flaschenhälse frühzeitig entdeckt und Lösung entwickelt werden.

#3 Kein Zugang zum Osmose-Frontend

So früh wie möglich den Zugriff beantragen. Ein öffentlicher Beta-Release und Beispiele der generierten Rapporte sollen Vertrauen schaffen und beweisen, dass die Qualität genügend ist.

#4 Manuelle Konfiguration bei jedem Lauf nötig

Früher technischer Durchstich. Konfigurationsmöglichkeiten anbieten, sodass das Tool unabhängig von der Infrastruktur laufen kann.

7.4.3 Eingetretene Risiken

Zum Zeitpunkt der Abgabe haben wir den Zugang zum Osmose-Frontend noch nicht erhalten. Jedoch ist die Ansprechperson der Osmose-Community vom Nutzen und der Qualität des Projektes überzeugt und hat grundsätzlich ein sehr positives Feedback gegeben.

Ansonsten sind keine Risiken eingetreten und das Projekt konnte erfolgreich abgeschlossen werden. Dazu beigetragen haben eine realistische Projektplanung und die Massnahmen zur Risikominimierung.

7.5 Zeiterfassung

Die Zeiten werden per GitLab Timetracking[41] erfasst. Für die Auswertung wird das Tool gtt[42] verwendet und mit Hilfe eines separaten Projektes einen Rapport für das Team und den Betreuer generiert.

7.5.1 Zeitauswertung pro Person

Die Tabelle 7.2 enthält die total für das Projekt aufgewendeten Stunden pro Person, sowie die prozentuale Abweichung zu der geforderten Zeit.

Name	Soll	Ist	Abweichung
Jari Elmer	360h	383h	6.4%
Timon Erhart	360h	381h	5.8%

Tabelle 7.2: Total aufgewendete Zeit pro Teilnehmer

7.5.2 Zeitauswertung pro Kategorie

Alle Arbeitspakete sind mit einer Kategorie versehen. So kann analysiert werden, für welche Tätigkeiten wie viel Zeit aufgewendet wird. In der Tabelle 7.3 können die dazugehörigen absoluten und relativen Werte nachgeschlagen werden.

Kategorie	Zeit	Anteil
Dokumentation	232h	31%
Implementation	381h	50%
Infrastruktur, CI/CD	10h	1%
Projektmanagement & Meetings	107h	14%
Technologische Einarbeitung	34h	4%

Tabelle 7.3: Zeit und Anteil pro Kategorie

7.6 Sitzungen

Es wurden zweiwöchentliche Sitzungen abgehalten. Dabei wurden die folgenden Punkte besprochen:

- Was wurde in der aktuellen Woche erreicht
- Gibt es Fragen/Unklarheiten
- Welche Tätigkeiten sind in der nächsten Woche zu erledigen

Die Sitzungsprotokolle werden den Betreuern in der Abgabe zur Verfügung gestellt.

ANHANG A

Aufgabenstellung

Aufgabenstellung: Qualitätsprüfung der Verknüpfungen von OpenStreetMap nach Wikidata

- Bachelorarbeit im Frühlingssemester 2022 Bachelor Informatik
- Autoren: Jari Elmer & Timon Erhart
- Betreuer: Prof. Stefan Keller, OST Campus Rapperswil Sascha Brawer, Programmierer im Ruhestand
- Industriepartner: SW-Communities u.a. von OpenStreetMap und Wikidata

Einleitung

OpenStreetMap (OSM) ist eine freie Landkarte der ganzen Welt mit rund 1 Milliarde geografischen Objekten. Die meisten Objekte in OpenStreetMap (rund 7 Milliarden) sind leere Knoten, die ausser ihrer geografischen Position keinerlei andere Information tragen, so etwa Gebäude-Ecken oder die Stützpunkte entlang der Mittellinien von Flüssen. Aber rund 184 Millionen Knoten (Nodes) tragen mindesten ein Tag, zudem gibt es 819 Millionen Linien (Ways) und 9 Millionen Relationen. Insgesamt umfasst OpenStreetMap also zurzeit rund eine Milliarde mehr oder weniger interessante Objekte. Wikidata ist eine freie Wissensdatenbank mit 96 Millionen Objekten. OpenStreetMap enthält vorwiegend geografische Informationen, zum Beispiel die Grundrisse von Häusern. Wikidata enthält vorwiegend Sachdaten, zum Beispiel wann ein Haus gebaut wurde, welches Wappen eine Stadt besitzt, oder nach wem eine Strasse benannt wurde.

OpenStreetMap verweist mit Tags auf Wikidata. Zum Beispiel zeigt OSM-Node 240062727 (Rapperswil) mittels dem Tag “wikidata” auf Wikidata-Objekt Q688539, wo unter anderem das Wappen der Stadt und die Bevölkerungsstatistik erfasst sind. OSM-Node 9275508333 (eine Gedenktafel am Pfrundhaus Rapperswil) zeigt auf Q109673464, wo erfasst ist, an wen diese Gedenktafel erinnert und von welchem Kunsthandwerker sie gestaltet wurde.

Mittels Kombination von OSM mit Wikidata lassen sich interessante Produkte bauen, zum Beispiel eine Karte von Schweizer Burgen mit Sachdaten aus Wikidata. Allerdings sind die meisten Wikidata-Tags in OpenStreetMap aktuell von Hand gesetzt und nicht wirklich geprüft. Deswegen ist die Abdeckung im Moment noch recht spärlich und die Qualität der Daten unbekannt.

Aufgabe

Nach einer kurzen Einarbeitung entwerfen und implementieren Sie ein Werkzeug, das OpenStreetMap mit Wikidata abgleicht. Ihr Werkzeug findet mögliche Fehler und Lücken in den existierenden Wikidata-Tags, generiert automatisch Vorschläge zur Korrektur und speist diese Vorschläge in eine existierende Fehlerdatenbank ein. Ziel der Arbeit ist, dass Ihr Werkzeug ein dauerhafter Teil der Infrastruktur zur Qualitätssicherung von OpenStreetMap wird. Ihr System muss daher mit den grossen Datenmengen der beiden Datenbanken zurechtkommen, ausgiebig getestet sein, und der Code muss in genügend hoher Qualität geschrieben werden, dass er auch von anderen verstanden und gewartet werden kann

Vorgehen

OST Campus Rapperswil

1. **Einarbeitung:** Sie machen sich mit OpenStreetMap und Wikidata vertraut. Sie machen von Hand ein paar Änderungen im iD-Editor von OpenStreetMap und gehen die Touren von Wikidata durch. Sie schreiben je ein Programm zum Einlesen der Datenbank-Dumps beider Projekte (61 GB komprimiert bei planet.openstreetmap.org; 71 GB bei dumps.wikimedia.org). Sie lernen Osmose kennen. Sie benutzen osm.wikidata.link und überfliegen dessen Quelltext. Sie überlegen sich, wie Sie die verfügbaren Daten indizieren wollen, um Ihr Werkzeug zu implementieren.
2. **Prototyp:** Sie implementieren Checks für ein paar wenige Fehlerkategorien, die besonders einfach zu finden sind. Beispiele: 1. OSM zeigt auf ein nicht existierendes Wikidata-Objekt; 2. OSM zeigt auf ein existierendes Wikidata-Objekt, das aber in Wikidata keine geografische Position hat; 3. OSM zeigt auf ein Wikidata-Objekt, dessen geografische Position weit entfernt von der Position in OSM ist. Ihr Prototyp generiert ein paar Korrekturvorschläge in einem XML-Format, das Sie von Hand in das “Issues Reporting API” von Osmose hochladen. Anschliessend arbeiten sie die hochgeladenen Korrekturvorschläge selber von Hand ab, damit Sie sehen, wie die Endnutzer:innen Ihr System verwenden.
3. **Realisierung:** Sie implementieren das System in einer Form, die auf Dauer in Produktion laufen kann. Implementieren ein einfaches Logging . Sie dokumentieren Ihr System so, dass auch andere es warten und erweitern können, und lancieren es unter einer freien Softwarelizenz.
4. **Ausbau:** Sie erweitern Ihr System um zusätzliche Funktionen. Beispiele: 1. OSM zeigt auf ein Wikidata-Objekt eines anderen Typs, etwa ein Brunnen auf eine Person. 2. OSM zeigt auf ein Wikidata-Objekt mit falschem Namen. 3. Ein OSM-Objekt trägt noch gar kein wikidata-Tag, aber in Wikidata findet sich eine Entsprechung, die von der geografischen Position, dem Namen und dem Typ her passen würde.

Lieferobjekte

1. Dokumentation, inkl. Textabstract (deutsch), Management Summary (deutsch), technischer Bericht und Software Engineering-Projekt (deutsch); Anhänge (Literaturverzeichnis, Inhalt).
2. Applikation «OSM Wikidata Quality Checker» als Docker Container auf Server unter Kontrolle des Betreuers.
3. Die vom Studiengang geforderten bzw. empfohlenen Lieferobjekte: Poster (nur digital), Broschüren-Abstract, kein Kurzvideo.

Vorgaben/Rahmenbedingungen

- Python, C++ oder Go.
- Docker
- Fehlerdatenbank: Osmose-Frontend
- SW-Entwicklung auf OST-Server
- Dokumentation: Latex

Vorgehen und Arbeitsweise: Die Studierenden wählen nach Rücksprache ein Vorgehensmodell zur Softwareentwicklung. Es gibt regelmässige Meetings mit vorbereiteten Unterlagen; wobei Ausnahmen vereinbart werden können.

Dokumentation

Zur Dokumentation (vgl. auch Lieferobjekte oben):

- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind (einheitliche Nummerierung).

OST
Campus
Rapperswil

- Die Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen (nicht ausschliesslich Wikipedia-Links auflisten).
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Eigenständigkeitserklärung, Nutzungsrechte) gemäss Vorgaben des Studiengangs und Absprache mit dem Betreuer.

Form der Dokumentation zuhanden Betreuer (Studiengang siehe separate Instruktionen):

- Bericht gebunden (1 Ex.)
- Alle Dokumente und Quellen der erstellten Software auf USB-Stick.

Bewertung

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Studienarbeit des Studiengangs Informatik mit besonderem Gewicht auf moderne Softwareentwicklung wie folgt:

- Projektorganisation (Gewichtung ca. 1/6)
- Bericht, Gliederung, Sprache (Gewichtung ca. 1/6)
- Inhalt inkl. Code (Gewichtung ca. 2/6)
- Gesamteindruck inkl. Kommunikation mit Externen (Gewichtung ca. 1/6)
- Mündliche Schlussprüfung (ca. 1/6)

Ein wichtiger Bestandteil der Arbeit ist, dass eine lauffähige, getestete Software abgeliefert wird (inkl. Softwaredokumentation).

Weitere Beteiligte

Mitarbeiter des Instituts für Software.

ANHANG B

Benutzerhandbuch (README)

Dies ist ein Auszug des `README.md` File aus dem Code-Repository. Die aktuelle Version kann auf dem öffentlichen GitLab-Repository[43] eingesehen werden.

README.md 13.61 KiB

OpenStreetMap Wikidata Quality Checker

Introduction

The `osm-wikidata-quality-checker` is a quality assurance tool for OpenStreetMap. It searches for invalid or inaccurate links between OpenStreetMap and Wikidata by evaluating Wikidata tags. The issues found are reported in different formats and can be sent to the Osmose frontend. For more details, see the System overview chapter.

The tool is written in Python and intended for use in a single Docker container.

Table of Contents

Installation

Prerequisite and system requirements

Since the tool is dockered, your system must have Docker installed and be connected to the Internet for dump downloads.

Your system should have at least:

- 16 GB RAM
- 300 GB free ROM
- 4 physical or logical CPU cores

Configuration

The tool can be configured by setting environment variables. Usually only the Osmose credentials need to be set. The others are useful for development or customization.

Environment variable	Default value	Description
`OSMOSE_SOURCE`		Osmose frontend id of source (username). *Mandatory*
`OSMOSE_CODE`		Osmose frontend password associated with the source. *Mandatory*
`OSMOSE_API_HOST`	`http://osmose.openstreetmap.fr/control/send-update`	Osmose frontend url. If left blank no report will be sent

Environment variable	Default value	Description
DO_DOWNLOAD	osm,wd,redirects	Specifies whether and which dumps the tool should download, disable all with no
DO_IMPORT	osm,wd,redirects	Specifies which databases the tool should create, disable all with no
DO_CHECKS	yes	Values 'no' does not execute any checks
OSM_DUMP_URL	https://planet.openstreetmap.org/pbf/planet-latest.osm.pbf	URL of the OpenStreetMap dumps to download
WD_DUMP_URL	https://dumps.wikimedia.org/wikidatawiki/entities/latest-all.json.bz2	URL of the Wikidata dumps to download
WD_REDIRECTS_URL	https://qrank.wmcloud.org/download/qredirects.csv.gz	URL of the Wikidata redirects to download
WD_DUMP_FILENAME	latest-all.json.bz2	Filename of the download OpenStreetMap dump
OSM_DUMP_FILENAME	planet-latest.osm.pbf	Filename of the download Wikidata dump
WD_REDIRECTS_FILENAME	qredirects.csv.gz	Filename of the download Wikidata redirects
OSM_DB_FILENAME	osm.db	Filename of the OpenStreetMap database
WD_DB_FILENAME	wikidata.db	Filename of the Wikidata database
MP_CPU_THRESHOLD	4	Threshold for the use of multiprocessing on checks

Run docker

Example of a minimal docker-compose file. The `outside_docker/` paths refer to a freely choosable directory on the system where outputs will be placed. The docker container will terminate after the script finishes.

```yaml
version: "3"

services:
  osm-wikidata-quality-checker:
    image: registry.gitlab.com/geometalab/osm-wikidata-quality-checker:latest
    restart: "no"
    environment:
      OSMOSE_SOURCE: "username"
      OSMOSE_CODE: "password"
    volumes:
      - /outside_docker/dumps:/data/dumps
      - /outside_docker/db:/data/db
      - /outside_docker/reports:/data/reports
      - /outside_docker/logs:/data/logs
```

Outputs

Reports

The following files are generated and saved in the `reports` folder. They contain all found issues of a run:

- `<date_time>_report.csv` : CSV file which contains all issues
- `<date_time>_report.xlsx` : Excel file which contains all issues
- `<date_time>_osmose_report.xml` : XML file which is sent to the Osmose frontend

> **Important**: At the moment, non-area `OsmRealtion` 's have no location. Issues detected on such objects are not beeing reported to Osmsose until we have a reliable and fast way to get the location of relations that are not areas.

Logs

Each run generates a log file inside of `logs` folder.

- `log_<date_time>.log`

Other artefacts

The files in the folders `dumps` and `db` are files that are rewritten with each run. They can be deleted between runs without any consequences, but may help with debugging and development.

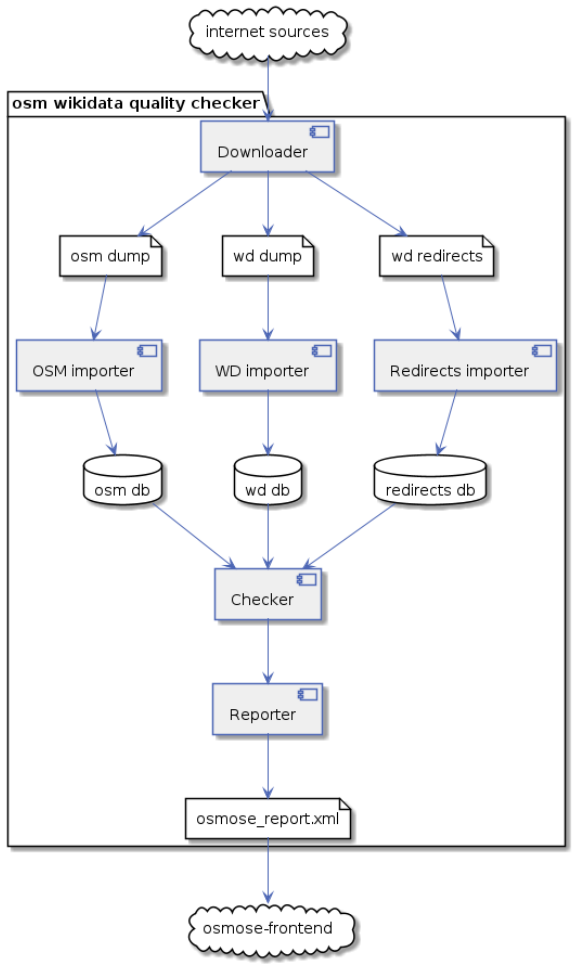
How the tools works

System overview

The graphic below gives an overview of the system and shows the data flow.

In a nutshell, the tool does the following:

1. Download dump files for OpenStreetMap and Wikidata.
2. Import relevant information from the dumps into local databases (indexes).
3. Perform checks on links from OSM to WD
4. Report the found issues to the Osmose frontend



Basic checks

These checks are always done and ensure, that a valid and existing Wikidata tag is present.

Check name	Description	Issue class
`_is_valid_format`	Checks if the value of any wikidata tag starts with letter Q followed by a integer (e.g. `Q123`). Multible values are allowed when seperated by a semicolon (e.g. `Q123;Q124`). Lexems (e.g. `L123`) are also a valid entry, but are not checked.	`InvalidWdItemFormatIssue`
part of `check` function	Checks if the linked item exists on Wikidata.	`NotExistingWdItemIssue`
`_is_redirected`	Checks if the wikidata item is redirected. It then offers a fix suggestion with the redirection target	`RedirectedWdItemIssue`

Link checks

After the basic checks, further checks are performed for each link. A link is a single connection between an OSM object and a wikidata element. For Example, an OSM node with the tags `wikidata=Q1;Q2` as well as `brand:wikidata=Q3` has 3 links.

The order of the checks is crucial. It stops at the first find, so at most one issue per link is generated.

Check name	Description	Issue class
`check_living_organism`	It is not allowed to link wikidata tags `wikidata`, `brand:wikidata`, `operator:wikidata`, `network:wikidata` or `flag:wikidata` directly to a living organism (human, animal, plant). The check proivde fixes for suitable secondary Wikidata links (e.g. `subject:wikidata` or `buried:wikidata`).	`LivingOrganismIssue`
`check_unpermitted_instance,`	This check looks for links on Wikidata items with prohibited categories. These are `Wikimedia disambiguation page (Q4167410)`, `Wikimedia category (Q4167836)`, `Wikimedia list article (Q13406463)` and `list (Q12139612)` According to Key:Wikidata, these are not permitetted, as only items that relate directly to the feature should be linked.	`UnpermittedInstanceIssue`
`check_primary_tag_instance_of`	Checks if any tags matches with the claims of the linked Wikidata item. E.g. an osm-object with a `industrial=port` tag should be linked to an wikidata item which is an `instance of (P31)` or `subclass of (P279)` of `Port (Q44782)` (or more specific).	`PrimaryTagClaimMismatchIssue`
`check_secondary_tags_instance_of`	If the OSM object has a secondary Wikidata link, the prefix should match the category of the Wikidata item. E.g. `brand:wikidata=` should be linked to a `instance of (P31)` or `subclass of (P279)` of `brand (Q98)`, `organization (Q43229)` or `trademark (Q167270)` (or more specific).	`SecondaryTagClaimMismatchIssue`
`check_places`	If an OSM object has a `place` tag, it checks if coordinates, name and postal_code match the linked Wikidata item. This is done by calculating a score over all existing features. (Currently, not all values for place are supported.)	`PlaceMismatchIssue`
`check_very_large_distance`	Checks if the coordinates in OpenStreetMap and Wikidata are very fare apart (>2000km), which is most likely incorrect.	`VeryLargeDistanceIssue`
`check_replaced_wikidata_item` *(inactive)*	Checks if the Wikidata item has a `replaced by (P1366)` property. It then offers a fix suggestion with the replaced target. The check is currently inactive due to lack of accuracy and could be improved.	`ReplacedWdItemIssue`

Check name	Description	Issue class
`check_dissolved_wd_item` *(inactive)*	Checks if the Wikidata item has a `dissolved, abolished or demolished date (P576)` property. The check is currently inactive due to lack of accuracy and could be improved.	`DissolvedWdItemIssue`

Authors

- Timon Erhart
- Jari Elmer

ANHANG C

Hinweise zu CI/CD

Es wird die Gitlab-CI verwendet, Listing C.1 zeigt die komplette Konfiguration. Es gibt drei verschiedene Stages, diese müssen ohne Fehler durchlaufen, damit die Pipeline den Status `success` erhält.

- **lint**: Code-Qualitätsprüfung bestehend `pylint` (linter), `black` (code formatting), `isort` (import order) und `mypy` (static type checker).
- **test**: Ausführen aller Unit-Test
- **publish**: Für die Branches `develop` und `master` wird ein Docker-Container gebaut und im GitLab-Repository publiziert.

Die Unit-Tests werden in Form eines JUnit Reports GitLab als Artefakt zur Verfügung gestellt (`report.xml`). So können bei einem Merge-Request gleich die Testresultate angeschaut werden. Bei der Ausführung der Unit-Tests wird gleichzeitig die Code-Coverage berechnet. Das Total der Coverage wird von GitLab aus dem Job-Log geparsed[44]. Für die Test-Coverage-Visualization[45] wird ein `coverage.xml` Artefakt erstellt.

```yaml
image: python:3.10

stages:
  - lint
  - test
  - publish

variables:
  TAG_SNAPSHOT: $CI_REGISTRY_IMAGE:snapshot
  TAG_TEST: $CI_REGISTRY_IMAGE:test
  TAG_LATEST: $CI_REGISTRY_IMAGE:latest
  TAG_BRANCH: "$CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG"

.before_script_poetry:
  before_script:
    - pip install poetry
    - poetry config virtualenvs.create false
    - poetry install

unit-tests:
  extends: .before_script_poetry
  stage: test
  needs: []
  script:
    - poetry run coverage run -m --source=osm_wikidata_quality_checker pytest --junitxml=report.xml tests
    - poetry run coverage xml
    - poetry run coverage report
  artifacts:
    when: always
    reports:
      junit: report.xml
      cobertura: coverage.xml
    paths:
      - ./tests/dumps/osm_import_runtime.log
      - ./tests/dumps/wd_import_runtime.log

black-code-formatter:
  extends: .before_script_poetry
  stage: lint
  needs: []
  allow_failure: false
  script:
    - poetry run black --check .

import-sort-check:
  extends: .before_script_poetry
  stage: lint
  needs: []
  allow_failure: false
  script:
    - poetry run isort -c .

linter-check:
  extends: .before_script_poetry
  stage: lint
  needs: []
  allow_failure: true
  script:
    - poetry run pylint osm_wikidata_quality_checker

static-type-check:
  extends: .before_script_poetry
  stage: lint
  needs: []
  allow_failure: false
  script:
    - poetry run mypy -p osm_wikidata_quality_checker

```

```yaml

publish-snapshot:
  image: docker:latest
  stage: publish
  needs: ["unit-tests", "static-type-check"]
  services:
    - docker:dind
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
  script:
    - docker build --pull -t "$TAG_SNAPSHOT" .
    - docker push "$TAG_SNAPSHOT"
  only:
    - develop

publish-latest:
  image: docker:latest
  stage: publish
  needs: ["unit-tests", "static-type-check"]
  services:
    - docker:dind
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
  script:
    - docker build --pull -t "$TAG_LATEST" .
    - docker push "$TAG_LATEST"
  only:
    - main
```

Listing C.1: GitLab CI Pipeline

ANHANG D

Codemetriken

Codezeilen

Die Codebasis umfasst **2’550** Codezeilen. Dazu kommen die Unit-Tests, welche weitere **1’128** Codezeilen umfassen.

Unit-Tests

Es existieren **64** Unit-Tests wobei **177** Asserts ausgeführt werden.

Pushed Commits auf den develop-Branch

Es wurden **634** Commits auf den develop-Branch gepushed. Da teilweise bei den Merge-Requests die Funktion ”Squash commits” verwendet wurde, kann davon ausgegangen werden, dass die wirkliche Anzahl an getätigten Commits noch um einiges höher liegt.

Merge Requests

Über die gesamte Projektdauer wurden **105** Merge Requests eröffnet und **100** davon gemergt.

ANHANG E

Testabdeckung

Der Coverage-Report in Tabelle E.1 kann mit den Befehlen in Listing E.1 erstellt werden.

```
poetry run coverage run -m --source=osm_wikidata_quality_checker pytest tests
poetry run coverage report
```

Listing E.1: Coverage Report generieren

Module	statements	missing	coverage
osm_wikidata_quality_checker/__init__.py	1	0	100%
osm_wikidata_quality_checker/__main__.py	6	6	0%
osm_wikidata_quality_checker/checker.py	74	21	72%
osm_wikidata_quality_checker/checker_issues.py	113	0	100%
osm_wikidata_quality_checker/checks_common.py	82	8	90%
osm_wikidata_quality_checker/database_common.py	87	4	95%
osm_wikidata_quality_checker/downloader.py	10	0	100%
osm_wikidata_quality_checker/env.py	42	0	100%
osm_wikidata_quality_checker/import_common.py	38	4	89%
osm_wikidata_quality_checker/link_checks.py	113	21	81%
osm_wikidata_quality_checker/main.py	133	105	21%
osm_wikidata_quality_checker/osm_database.py	216	41	81%
osm_wikidata_quality_checker/osm_import.py	137	26	81%
osm_wikidata_quality_checker/osmose_reporter.py	155	27	83%
osm_wikidata_quality_checker/reporter.py	126	26	79%
osm_wikidata_quality_checker/wd_constants.py	106	0	100%
osm_wikidata_quality_checker/wd_database.py	67	4	94%
osm_wikidata_quality_checker/wd_import.py	65	4	94%
osm_wikidata_quality_checker/wd_redirects_database.py	26	1	96%
osm_wikidata_quality_checker/wd_redirects_import.py	17	0	100%
TOTAL	**1614**	**298**	**82%**

Tabelle E.1: Testabdeckung aus coverage.py v6.3.2, vom 13.06.2022

ANHANG F

XML Osmose Report Example

Listing F.1 zeigt ein Beispiel eines generierten Osmose-Reports.

```xml
<?xml version="1.0" encoding="utf8"?>
<analysers timestamp="2022-05-18T11:33:04Z">
  <analyser timestamp="2022-05-18T11:33:04Z">
    <class item="3031" tag="wikidata, tag, fixme" id="3" level="2"
    ↪  source="https://gitlab.com/geometalab/osm-wikidata-quality-checker">
      <classtext lang="en" title="Redirected value for Wikidata tag"/>
      <detail lang="en" title="This Wikidata item is a redirect, you shoud change the QID to the redirect
      ↪  destionation."/>
      <fix lang="en" title="Change the value of the affected tag to the suggested new Wikidata-ID."/>
    </class>
    <class item="3031" tag="wikidata, tag, fixme" id="2" level="2"
    ↪  source="https://gitlab.com/geometalab/osm-wikidata-quality-checker">
      <classtext lang="en" title="Wikidata item does not exist"/>
      <detail lang="en" title="The linked Wikidata item was not found in the Wikidata database dump."/>
      <fix lang="en" title="Check if the tag realy does not exists, then delete the affected tag."/>
      <trap lang="en" title="Because the used OSM database dump can be newer than the Wikidata database
      ↪  dump, it is possible that false positives are generated for very new items. Therefore, always
      ↪  make sure that it really does not exist."/>
    </class>
    <error class="3" subclass="388672817">
      <way id="12618874" user="captaininler" version="25" lat="" lon="">
        <tag k="wikidata" v="Q27329407"/>
      </way>
      <text lang="en" value="Wikidata item is redirected to Q1620404."/>
      <fixes>
        <fix>
          <way id="12618874">
            <tag k="wikidata" action="modify" v="1620404"/>
          </way>
        </fix>
      </fixes>
    </error>
    <error class="2" subclass="1673010365">
      <way id="23687791" user="chatelao" version="14" lat="" lon="">
        <tag k="name:etymology:wikidata" v="Q106676558;Q106676550"/>
      </way>
      <text lang="en" value="The linked Wikidata item Q106676558 does not exists on Wikidata"/>
      <fixes>
        <fix>
          <way id="23687791">
            <tag k="name:etymology:wikidata" action="delete"/>
          </way>
        </fix>
      </fixes>
    </error>
  </analyser>
</analysers>
```

Listing F.1: Osmose Report Example

Literaturverzeichnis

Basiri, A., Haklay, M., Foody, G. & Mooney, P. (2019). Crowdsourced geospatial data quality: challenges and future directions. *International Journal of Geographical Information Science*, *33* (8), 1588-1593. Zugriff auf https://doi.org/10.1080/13658816.2019.1593422 doi: 10.1080/13658816.2019.1593422

Brawer, S., User:Infrastruktur & User:Dipsacus fullonum. (2022, März). *Wikidata project chat: Machine-readable list of all wikidata redirects?* Zugriff auf https://www.wikidata.org/wiki/Wikidata:Project_chat/Archive/2022/03#Machine-readable_list_of_all_Wikidata_redirects?

Dasanayake, S., Markkula, J., Aaramaa, S. & Oivo, M. (2015). Software architecture decision-making practices and challenges: An industrial case study. In *2015 24th australasian software engineering conference* (S. 88-97). doi: 10.1109/ASWEC.2015.20

Gamma, E., Helm, R., Johnson, R., Johnson, R. E. & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.

Hoch, M. (2014, November). *Wikimedia forum: json dumps have duplicate items (one for the redirect, one for the target).* Zugriff auf https://phabricator.wikimedia.org/T74678 (Kommentar von User hoo (Marius Hoch) vom Nov 15 2014, 4:50 PM)

Hoffmann, S. (2022, Mai). *Pyosmium area datatype documentation.* Zugriff auf https://docs.osmcode.org/pyosmium/latest/ref_osm.html#osmium.osm.Area

Mediawiki. (2017, June). *Osmgadget.* Zugriff auf https://www.mediawiki.org/wiki/User:Bjohas/OSMgadget

Open Knowledge Foundation. (2022, Mai). *Was ist open data?* Zugriff auf https://opendatahandbook.org/guide/de/what-is-open-data/

OpenStreetMap. (2020a, Oktober). *Wikipedia link improvement project.* Zugriff auf https://wiki.openstreetmap.org/wiki/Wikipedia_Link_Improvement_Project#Missing_Wikidata_tags

OpenStreetMap. (2020b, August). *Wiwosm.* Zugriff auf https://wiki.openstreetmap.org/wiki/WIWOSM

OpenStreetMap. (2021a, Februar). *Map features.* Zugriff auf https://wiki.openstreetmap.org/wiki/Map_features

OpenStreetMap. (2021b, Februar). *Namespace.* Zugriff auf https://wiki.openstreetmap.org/wiki/Namespace

OpenStreetMap. (2021c, August). *Osm - wikidata matcher.* Zugriff auf https://wiki.openstreetmap.org/wiki/OSM_%E2%86%94_Wikidata_matcher

OpenStreetMap. (2021d, Mai). *Postgresql.* Zugriff auf https://wiki.openstreetmap.org/wiki/DE:PostgreSQL

OpenStreetMap. (2021e, Oktober). *Tag:place=village.* Zugriff auf https://wiki.openstreetmap.org/wiki/Tag:place%3Dvillage

OpenStreetMap. (2022, April). *Good practice.* Zugriff auf https://wiki.openstreetmap.org/wiki/Good_practice

OpenStreetMap. (2022a, März). *Key:building.* Zugriff auf https://wiki.openstreetmap.org/wiki/Key:building

OpenStreetMap. (2022b, Mai). *Osm key:name:etymology.* Zugriff auf https://wiki.openstreetmap.org/wiki/Key:name:etymology#Rationale

OpenStreetMap. (2022c, Mai). *Osm key:wikidata.* Zugriff auf https://wiki.openstreetmap.org/wiki/Key:wikidata

OpenStreetMap. (2022d, Februar). *Osmose.* Zugriff auf https://wiki.openstreetmap.org/wiki/Osmose

OpenStreetMap. (2022e, April). *Planet.osm.* Zugriff auf https://wiki.openstreetmap.org/wiki/Planet.osm

OpenStreetMap. (2022f, März). *Relation:multipolygon.* Zugriff auf https://wiki.openstreetmap.org/w/index.php?title=Relation:multipolygon&uselang=de

OpenStreetMap. (2022g, Januar). *Sophox.* Zugriff auf https://wiki.openstreetmap.org/wiki/Sophox

OpenStreetMap. (2022h, März). *Wikidata linking from wikidata to osm.* Zugriff auf https://wiki.openstreetmap.org/wiki/Wikidata#Linking_from_Wikidata_to_OSM

OpenStreetMap. (2022i, März). *Wikidata tools.* Zugriff auf https://wiki.openstreetmap.org/wiki/Wikidata#Tools

Pidoux, E. (2014). *Git best practices guide*. Packt Publishing Ltd.

See, L., Mooney, P., Foody, G., Bastin, L., Comber, A., Estima, J., ... Rutzinger, M. (2016). Crowdsourcing, citizen science or volunteered geographic information? the current state of crowdsourced geographic information. *ISPRS International Journal of Geo-Information*, *5* (5). Zugriff auf https://www.mdpi.com/2220-9964/5/5/55 doi: 10.3390/ijgi5050055

Wikidata. (2022a, Januar). *Linking from wikidata to osm.* Zugriff auf https://www.wikidata.org/wiki/Wikidata:OpenStreetMap#Linking_from_Wikidata_to_OSM

Wikidata. (2022b, Mai). *Statistics - wikidata.* Zugriff auf https://www.wikidata.org/wiki/Special:Statistics

Wikipedia. (2022a, Februar). *Bahnhof Berlin Pichelsberg.* Zugriff auf https://de.wikipedia.org/wiki/Bahnhof_Berlin_Pichelsberg

Wikipedia. (2022b, Februar). *OpenStreetMap.* Zugriff auf https://de.wikipedia.org/wiki/OpenStreetMap

Zimmermann, O. (2020, Apr). *Architectural decisions - the making of.* Zugriff auf https://ozimmer.ch/practices/2020/04/27/ArchitectureDecisionMaking.html

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Anmerkungen

1. https://www.wikidata.org/wiki/Wikidata:Statistics/de
2. https://gitlab.ost.ch/ba_osm_wikidata/osm-wikidata-quality-checker
3. https://gitlab.com/geometalab/osm-wikidata-quality-checker
4. https://www.openstreetmap.org/
5. https://castle-map.infs.ch
6. https://wiki.openstreetmap.org/wiki/OSM_XML
7. https://wiki.openstreetmap.org/wiki/Proposed_features/Wikidata
8. https://www.wikidata.org/
9. https://query.wikidata.org/
10. http://osmose.openstreetmap.fr
11. https://wiki.openstreetmap.org/wiki/Osmose#Issues_file_format
12. https://github.com/osm-fr/osmose-backend/blob/master/doc/0-Index.md#classes-of-issues
13. https://osm.wikidata.link/
14. https://sophox.org/
15. https://wiwosm.toolforge.org/osm-on-ol/kml-on-ol.php
16. https://wiki.openstreetmap.org/wiki/Wikipedia_Link_Improvement_Project
17. https://www.mediawiki.org/wiki/User:Bjohas/OSMgadget
18. https://github.com/osmlab/wikidata-osm
19. https://osmlab.github.io/wikidata-osm/
20. https://www.wikidata.org/wiki/Wikidata:Lexicographical_data
21. https://taginfo.openstreetmap.org/keys/building#values
22. https://taginfo.openstreetmap.org/search?q=place
23. https://de.wikipedia.org/wiki/Bahnstrecke_Bad_M%C3%BCnder%E2%80%93Bad_Nenndorf
24. https://github.com/osm-fr/osmose-backend/blob/master/osmose_config.py
25. https://osmcode.org/pyosmium/
26. https://osmcode.org/libosmium/
27. https://github.com/shapely/shapely
28. https://github.com/dahlia/wikidata
29. https://www.sqlalchemy.org/
30. https://pypi.org/project/fuzzywuzzy/
31. https://docs.pytest.org/
32. https://github.com/nedbat/coveragepy
33. http://osmose.openstreetmap.fr/de/issues/open?source=10349
34. https://github.com/brawer/osmviews-py
35. https://github.com/brawer/wikidata-qrank
36. https://www.python.org/dev/peps/pep-0008/
37. https://github.com/psf/black
38. https://pycqa.github.io/isort/
39. https://pylint.org/
40. http://mypy-lang.org/
41. https://docs.gitlab.com/ee/user/project/time_tracking.html
42. https://github.com/kriskbx/gitlab-time-tracker
43. https://gitlab.com/geometalab/osm-wikidata-quality-checker
44. https://docs.gitlab.com/ee/ci/pipelines/settings.html#add-test-coverage-results-to-a-merge-request
45. https://docs.gitlab.com/ee/user/project/merge_requests/test_coverage_visualization.html