ODDToolkit: An Open-Source Toolkit to Help Bridge the Developer Gap in Ontology Driven Design

Maxim Van de Wynckel[1,*], Geert Van Haute[1]

[1]*Department of Environment and Spatial Development, Flemish Government, Belgium*

Abstract

Linked open data remains difficult to operationalise for software teams when OWL-based models are documented primarily for ontology engineers rather than for implementers. Furthermore, maintaining documentation for implementers becomes time consuming when the ontology is under development. We present *ODDToolkit*, an open-source Ontology Driven Design toolkit that helps generate developer-facing artefacts from three inputs: an OWL ontology, a SKOS concept scheme that provides business-specific names, and a configuration file for developer-specific design choices. The toolkit emits SQL DDL, Java and TypeScript models, SHACL shapes, diagrams, and documentation. Our design goal is not to fully automate translation of arbitrary vocabularies, but to achieve repeatable regeneration of implementation artefacts as an ontology evolves. We summarise the pipeline and report its use in a Flemish government environmental reporting project, where the generated artefacts supported implementation, documentation, and validation around a shared ontology as it evolves.

Keywords

ontology driven design, documentation, code generation, data modelling, software development

1. Introduction and Related Work

Linked Open Usable Data stresses that linked data should be consumable by developers [1]. However, in practice, software development teams still struggle to bridge ontology engineering and software delivery [2, 3]. The documentation burden described in NeOn [4] and later industrial ontology-engineering work [5] becomes even more challenging when the ontology, requirements, and the codebase all evolve together.

Existing tooling only partly covers this gap. Widoco, Parrot, and OnToology generate ontology-centric documentation and evaluation reports [6–8]; other approaches derive specific artefacts such as UML views [9], object-oriented libraries [10], REST APIs [11], or SHACL shapes [12]. Broader schema and model-transformation ecosystems, such as the Flemish OSLO toolchain, also support model-to-artefact workflows [13–15]. What this tooling landscape does not address is not a single missing generator, but the consistent alignment of ontology semantics, naming conventions [16, 17], and implementation constraints across several developer-facing artefacts. Our solution addresses this by keeping a governed OWL ontology as the formal source, using SKOS to add project-specific developer names, and projecting one enriched representation to code, SQL, SHACL, diagrams, and documentation in the same regeneration step. Developer

Posters, Demos, Blue Sky, and Tutorials at SEMANTiCS 2026, Sep 2026, Ghent, Belgium

EMAIL: maxim.vandewynckel@vlaanderen.be (M. Van de Wynckel)

ORCID: 0000-0003-0314-7107 (M. Van de Wynckel); 0009-0002-9836-8139 (G. Van Haute)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

decisions that are not expressed in the ontology are configured in a single configuration file so the output remains consistent with the codebase.

Following the ontology-driven design paradigm [18], ODDToolkit was built for projects where an ontology must stay aligned with developer-facing artefacts (e.g., to prevent unnecessary transformations or inconsistencies in the application data model and publication data model). It addresses three practical needs: a naming layer that projects ontology predicates to implementation names, a shared pipeline that derives multiple artefacts from one enriched model, and integration with normal build workflows so regeneration happens together with ontology changes. The toolkit targets governed ontologies rather than arbitrary third-party vocabularies and assumes project conventions such as SKOS labels, Hydra IRI templates, and explicit enough relation semantics for downstream generation.

This scope also implies a number of structural assumptions. ODDToolkit is aimed at ontologies under active governance, where a project can provide the necessary information to derive cardinality constraints, naming and relation semantics. The toolkit therefore prioritises implementation alignment and regeneration within a controlled development process over full automation for arbitrary third-party vocabularies.

2. Solution and Implementation

ODDToolkit[1] is a standalone CLI that combines an OWL ontology [19], a SKOS concept scheme [20], and a configuration file capturing the architectural choices of the software development teams (e.g., handling primary keys and cardinality). The ontology remains the formal source model, the SKOS layer provides business or developer-facing names, and all artefacts are derived from one enriched intermediate representation.

As shown in Figure 1, the pipeline has two phases. In *preparation*, the toolkit resolves imports, caches external vocabularies, and applies OWL reasoning with Apache Jena [21]. It then extracts classes and properties, derives keys from Hydra IRI templates [22], interprets cardinality constraints from `owl:Restriction` into a $[n_{\min}..n_{\max}]$ interval per property, and propagates inverse-property information so relation multiplicities can be reused across generators: given forward and inverse intervals $[a..b]$ and $[c..d]$, the relation is classified as 1:1, 1:n, m:1, or m:n depending on whether b and d exceed one, which then drives foreign-key placement and join-table generation. The SKOS mapping is applied at this stage, allowing the same ontology predicate to be exposed with developer-oriented names that better match object-oriented and relational conventions [16, 17]. Table 1 summarises how OWL constructs map to concepts across all generated artefacts.

For example, an OWL class `:EmissionPoint` can use a Hydra template that specifies which predicates are used within the URI. ODDToolkit then derives the same composite identity for SQL, while SKOS can rename predicates such as `dct:issued` to `validFrom` and `sosa:isHostedBy` to `location`. Cardinality constraints can also be applied: a `[1..1]` restriction becomes a required field and a `NOT NULL` column while `[*..*]` relations result in join tables.

This also yields a concrete implementation projection. In SQL, the generated table uses a composite primary key over `uuid` and `valid_from`, with a separate identity table anchoring

[1] https://github.com/milieuinfo/oddtoolkit

Table 1
Mapping from OWL/vocabulary constructs to generated artefact concepts.

Input construct	Detail	Generated concept
`owl:Class`		Class / Interface / SQL table / `sh:NodeShape`
`owl:Restriction`	$[n_{\min}..n_{\max}]$ interval	Nullable field / collection type / `NOT NULL`
`hydra:IriTemplate`	variable mapping	Primary key (`@Id` / `PRIMARY KEY`)
`owl:inverseOf`	inferred	$m:n$ join table / $m:1$ FK / $1:1$ field
`skos:Concept`	`owl:equivalentProperty`	Field / column / property name
`rdfs:subClassOf`	hierarchy	Inheritance / interface implementation

foreign keys to the non-temporal `uuid` so entity identity stays decoupled from entity state under this versioning pattern; in Java, the same pair becomes the JPA identity while `status` remains mandatory and `geometry` nullable. The same class is also projected to TypeScript and SHACL, so the developer-facing field names and structural constraints stay aligned across code, schema, and validation artefacts.

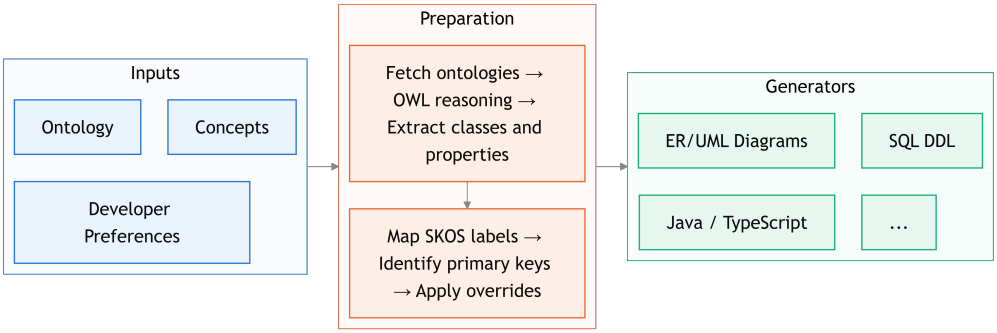

Figure 1: The ODDToolkit processing pipeline.

In *generation*, the enriched model is projected to multiple outputs. The SQL generator turns classes into tables, derives composite keys from Hydra variables, and places foreign keys or join tables from resolved multiplicities; multiple $m:n$ relations between the same table pair are merged into one join table with an auto-generated discriminator column, avoiding redundant tables. The Java generator emits JPA-oriented entities, the TypeScript generator produces JSON-LD-oriented typed models [23], the SHACL generator emits validation constraints from the same structural rules [24, 25], and the Bikeshed generator creates specification source files that complement ontology-centric documentation such as Widoco [6, 26]. UML and ER diagrams are generated from the same model, so decisions propagate to both textual and visual outputs that can immediately be used in communication with software teams. Changes to the ontology will automatically propagate to all outputs. Generated artefacts are deterministically sorted and arranged to enable support version control to track changes in the generated artefacts. This gives developers a clear view of what changed in the underlying ontology and SKOS mapping, and how that affects the implementation artefacts.

The Java output also preserves ontology-driven polymorphism where it matters. If an inherited parent belongs to an external vocabulary namespace and is actually reused as a

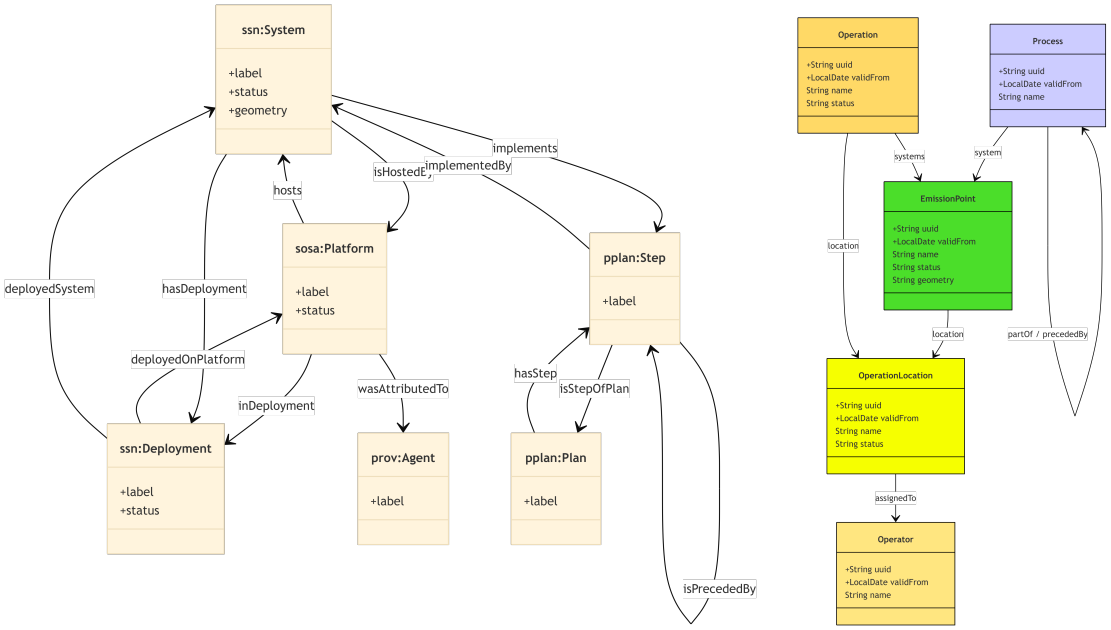

Figure 2: RDF/SOSA/SSN/P-PLAN model (left) and ODDToolkit-generated class diagram (right) for the IED ontology.

relation target by multiple in-scope classes, ODDToolkit emits it as an interface rather than a concrete class; this is why external concepts such as `ssn:System` can surface as reusable interface types in generated code instead of being flattened away.

For these diagrams, OWL class hierarchies are rendered as inheritance arrows; object properties become directed associations labelled with the SKOS-derived name. As illustrated in Figure 2, association labels derive from the SKOS concept scheme (e.g. `sosa:isHostedBy` → *location*, `prov:wasAttributedTo` → *assignedTo*, `pplan:isStepOfPlan` and `pplan:isPrecededBy` → *partOf* / *precededBy*).

The main implementation choice is architectural rather than format-specific: all generators consume the same prepared model instead of reinterpreting the ontology independently. This keeps code, schema, validation artefacts, and documentation aligned while allowing new generators to be added without changing the preparation stage.

In addition to the generated artefacts, ODDToolkit can include traceability information in the output. By automatically enriching existing software artefacts with links to the underlying linked data classes and properties, developers can navigate from code to the ontology and back. This is particularly useful for onboarding new team members or for maintaining the system as the ontology evolves.

3. Use Case and Observations

We applied ODDToolkit within the ontology engineering lifecycle of a Flemish government project for environmental reporting. In earlier projects, ontologies used for publication were

```sql
CREATE TABLE emission_point_identity (
  uuid VARCHAR PRIMARY KEY
);
CREATE TABLE emission_point (
  uuid       VARCHAR NOT NULL,
  valid_from DATE    NOT NULL,
  status     VARCHAR NOT NULL,
  geometry   VARCHAR,
  PRIMARY KEY (uuid, valid_from),
  FOREIGN KEY (uuid)
    REFERENCES emission_point_identity(uuid)
);
```

```java
@Entity
public class EmissionPoint
    implements ISystem {

  @Id private String uuid;
  @Id private LocalDate validFrom;

  @Column(nullable = false)
  private Status status;
  private String geometry;

  // ...
}
```

Figure 3: Generated SQL DDL (left) and JPA entity (right) for `:EmissionPoint`.

often retrofitted to data models that had already been shaped by implementation decisions. For the IED (Industrial Emissions Directive) and the IEPR regulation [27], we instead started from a new ontology intended to model industrial processes and their emissions, with implementation artefacts generated around that conceptual model. This change in approach required a new workflow for ontology engineers and developers to understand each other's artefacts.

The ontology reused SOSA [28], P-PLAN [29], PROV-O, and GeoSPARQL, which provided an interoperable modelling basis but also introduced abstractions and predicate names that were not immediately aligned with the development team's implementation vocabulary, a challenge consistent with prior observations about linked data adoption [2, 3]. This was particularly visible for developers who needed to understand the IED process model from technology-facing artefacts rather than from ontology axioms directly. Furthermore, developers were not familiar with the open-world assumption underlying linked data, in contrast to the closed-world assumption of the databases and programming languages they normally work with, resulting in a preference for more traditional artefacts.

The ontology, SKOS concept scheme, and configuration file were maintained in version control, with the generation pipeline executed in continuous integration so artefacts were regenerated and communicated on each ontology change. The SKOS concept scheme was jointly authored by ontology engineers and lead developers, capturing naming decisions that could not be inferred from the ontology alone; for instance, `sosa:isHostedBy` was mapped to `location` because the only relevant hosting relationship for an emission point in this domain is between the emission point and its operation location. Design decisions such as the use of surrogate keys for versioning were captured in the configuration file.

In practice, the generated Java and TypeScript models, as well as the SQL DDL, were used as implementation starting points. The diagrams supported project documentation, and the SHACL shapes fed validation and synthetic-data workflows. While the project is ongoing, the generated artefacts serve as technology-familiar entry points to the IEPR data model, complementing the formal ontology definitions rather than requiring developers to start from ontology axioms alone. This observation is not based on a comparative evaluation or developer interviews, but it illustrates how ODDToolkit can keep implementation, documentation, and validation artefacts aligned around a shared ontology.

4. Conclusion and Future Work

ODDToolkit presents a practical Ontology Driven Design pipeline that keeps documentation, validation artefacts, typed models, and schema-oriented outputs aligned around a shared enriched representation. The environmental reporting project in Section 3 shows that this workflow can be integrated into a development process and can provide useful implementation, documentation, and validation artefacts from one source.

The paper does not yet offer a quantitative evaluation of developer productivity, maintenance cost, or long-term evolution across multiple deployments. However, it demonstrates that good ontology engineering practices can be leveraged to aid in the generation and upkeep of documentation that helps developers understand and work with the underlying data models.

Our approach still depends on explicit project conventions, especially SKOS-based naming and Hydra-based key identification, so its strongest fit remains governed ontologies under active development. It does not yet solve preservation of customised generated code, migration of populated databases, preservation of derived indexes or summaries, or partial-update logic. Future work should therefore focus on interoperability with broader schema ecosystems and a wider evaluation across projects.

Supplemental Material. ODDToolkit is available on GitHub, including code and website documentation [30].

Declaration on Generative AI. During the creation of ODDToolkit, the authors used GitHub Copilot to help with the documentation and development. The authors take full responsibility for the content in the repository.

References

[1] D. Newbury, LOUD: Linked Open Usable Data and linked.art, in: Proceedings of the 2018 CIDOC Conference, 2018, pp. 1–11.

[2] J. Conde, A. Munoz-Arcentales, J. Choque, G. Huecas, Á. Alonso, Overcoming the Barriers of Using Linked Open Data in Smart City Applications, Computer 55 (2022) 109–118. doi:10.1109/MC.2022.3206144.

[3] C. Avila-Garzon, Applications, Methodologies, and Technologies for Linked Open Data: A Systematic Literature Review, International Journal on Semantic Web and Information Systems (IJSWIS) 16 (2020) 53–69.

[4] M. C. Suárez-Figueroa, A. Gómez-Pérez, M. Fernández-López, The NeOn Methodology for Ontology Engineering, in: M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, A. Gangemi (Eds.), Ontology Engineering in a Networked World, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 9–34. doi:10.1007/978-3-642-24794-1_2.

[5] M. Poveda-Villalón, A. Fernández-Izquierdo, M. Fernández-López, R. García-Castro, LOT: An Industrial Oriented Ontology Engineering Framework, Engineering Applications of Artificial Intelligence 111 (2022) 104755. doi:10.1016/j.engappai.2022.104755.

[6] D. Garijo, WIDOCO: A wizard for documenting ontologies, in: Proceedings of the International Semantic Web Conference (ISWC 2017), Springer, 2017, pp. 94–102.
[7] C. Tejo-Alonso, D. Berrueta, L. Polo, S. Fernández, Metadata for web ontologies and rules: Current practices and perspectives, in: E. García-Barriocanal, Z. Cebeci, M. C. Okur, A. Öztürk (Eds.), Proceedings of the International Conference on Metadata and Semantic Research (MTSR 2011), Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 56–67.
[8] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. Santana-Pérez, A. Fernández-Izquierdo, O. Corcho, Automating ontology engineering support activities with OnToology, Journal of Web Semantics (2018).
[9] K. Robles, A. Fraga, J. Morato, J. Llorens, Towards an Ontology-based Retrieval of UML Class Diagrams, Information and Software Technology 54 (2012) 72–86. doi:10.1016/j.infsof.2011.07.003.
[10] C. El Kaed, A. Ponnouradjane, A Model Driven Approach Accelerating Ontology-based IoT Applications Development, in: Proceedings of the SEMANTiCS 2017 Workshops (SIS-IoT), volume 2063 of *CEUR Workshop Proceedings*, 2017, pp. 1–8. URL: https://ceur-ws.org/Vol-2063/sisiot-paper4.pdf.
[11] P. Espinoza-Arias, D. Garijo, Ó. Corcho, Extending Ontology Engineering Practices to Facilitate Application Development, in: Proceedings of the 23rd International Conference on Knowledge Engineering and Knowledge Management (EKAW 2022), Springer, 2022, pp. 19–35. doi:10.1007/978-3-031-17105-5_2.
[12] A. Cimmino, A. Fernández-Izquierdo, R. García-Castro, Astrea: Automatic generation of SHACL shapes from ontologies, in: Proceedings of the 17th European Semantic Web Conference (ESWC 2020), Springer, Cham, 2020, pp. 497–513. doi:10.1007/978-3-030-49461-2_29.
[13] S. A. T. Moxon, H. Solbrig, N. L. Harris, P. Kalita, M. A. Miller, S. Patil, K. Schaper, C. Bizon, J. H. Caufield, S. C. Cuesta, C. Cox, F. Dekervel, D. M. Dooley, W. D. Duncan, T. Fliss, S. Gehrke, A. S. L. Graefe, H. Hegde, A. J. Ireland, J. O. B. Jacobsen, M. Krishnamurthy, C. Kroll, D. Linke, R. Ly, N. Matentzoglu, J. A. Overton, J. L. Saunders, D. R. Unni, G. Vaidya, W.-M. A. M. Vierdag, LinkML Community Contributors, O. Ruebel, C. G. Chute, M. H. Brush, M. A. Haendel, C. J. Mungall, LinkML: An Open Data Modeling Framework, GigaScience 15 (2026) giaf152. doi:10.1093/gigascience/giaf152.
[14] Digitaal Vlaanderen, OSLO UML Transformer, https://github.com/Informatievlaanderen/OSLO-UML-Transformer, 2026.
[15] Publications Office of the European Union, model2owl: Transform a UML model into a formal OWL ontology and SHACL shape, https://github.com/OP-TED/model2owl, 2026.
[16] S. Butler, M. Wermelinger, Y. Yu, Investigating Naming Convention Adherence in Java References, in: Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2015, pp. 41–50. doi:10.1109/ICSM.2015.7332450.
[17] A. Papamichail, A. V. Zarras, P. Vassiliadis, Do People Use Naming Conventions in SQL Programming?, in: A. Chatzigeorgiou, R. Dondi, H. Herodotou, C. Kapoutsis, Y. Manolopoulos, G. A. Papadopoulos, F. Sikora (Eds.), Proceedings of SOFSEM 2020: Theory and Practice of Computer Science, Springer International Publishing, Cham, 2020, pp. 429–440.
[18] P. Křemen, Z. Kouba, Ontology-driven information system design, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42 (2012) 334–344.

doi:10.1109/TSMCC.2011.2163934.

[19] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL 2: The Next Step for OWL, Journal of Web Semantics 6 (2008) 309–322. doi:10.1016/j.websem.2008.05.001, semantic Web Challenge 2006/2007.

[20] A. Miles, S. Bechhofer, SKOS Simple Knowledge Organization System Reference, W3C Recommendation, World Wide Web Consortium, 2009. URL: https://www.w3.org/TR/skos-reference/.

[21] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, K. Seaborne, K. Wilkinson, Jena: Implementing the Semantic Web Recommendations, in: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, WWW Alt. '04, Association for Computing Machinery, New York, NY, USA, 2004, pp. 74–83. doi:10.1145/1013367.1013381.

[22] M. Lanthaler, C. Gütl, Hydra: A Vocabulary for Hypermedia-Driven Web APIs, in: Proceedings of the 6th Workshop on Linked Data on the Web (LDOW 2013), volume 996 of *CEUR Workshop Proceedings*, 2013, pp. 35–38.

[23] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P.-A. Champin, N. Lindström, JSON-LD 1.1: A JSON-based Serialization for Linked Data, W3C Recommendation, World Wide Web Consortium, 2020. URL: https://www.w3.org/TR/json-ld11/.

[24] H. Knublauch, D. Kontokostas, Shapes Constraint Language (SHACL), W3C Recommendation, World Wide Web Consortium, 2017. URL: https://www.w3.org/TR/shacl/.

[25] K. Rabbani, M. Lissandrini, K. Hose, SHACL and ShEx in the Wild: A Community Survey on Validating Shapes Generation and Adoption, in: Proceedings of the ACM Web Conference 2022 Companion (WWW 2022), ACM, 2022, pp. 260–263. doi:10.1145/3487553.3524253.

[26] T. A. Jr., Bikeshed: A Spec Pre-Processor, https://tabatkins.github.io/bikeshed/, 2024. Accessed: 2026.

[27] European Parliament and Council of the European Union, Regulation (eu) 2024/1244 of the european parliament and of the council of 24 april 2024 on reporting of environmental data from industrial installations, establishing an industrial emissions portal and repealing regulation (ec) no 166/2006, https://eur-lex.europa.eu/eli/reg/2024/1244/oj/eng, 2024. Official Journal of the European Union, OJ L, 2024/1244, 2 May 2024.

[28] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, M. Lefrançois, SOSA: A Lightweight Ontology for Sensors, Observations, Samples, and Actuators, Journal of Web Semantics 56 (2019). doi:10.1016/j.websem.2018.06.003.

[29] D. Garijo, Y. Gil, Augmenting PROV with Plans in P-PLAN: Scientific Processes as Linked Data, in: Proceedings of the 2nd International Workshop on Linked Science (LISC 2012), volume 951 of *CEUR Workshop Proceedings*, 2012, pp. 1–4.

[30] M. Van de Wynckel, G. Van Haute, ODDToolkit - GitHub Repository, 2026. doi:10.5281/zenodo.21489607.