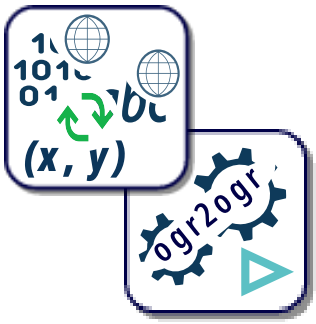

18.12.2025 - Semester Thesis Fall Semester 2025

Apache Hop Plugins

Implementation of two Plugins for GIS File Formats

Students: **Tobias Keel** tobias.keel@ost.ch
Simon Weibel simon.weibel@ost.ch

Advisor: **Prof. Stefan F. Keller**
stefan.keller@ost.ch

Abstract

Apache Hop is a visual data orchestration and enterprise data integration platform for designing, running, and monitoring data pipelines and workflows without writing code. It has been under continuous development for 25 years and is written in Java under an open-source licence. Users define a sequence of modular transforms in so-called pipelines, which are executed row by row and can run in parallel. These transforms range from generating sample data to executing external processes, enabling the flexible construction of reusable data processes. Out of the box, Hop does not support geospatial data processing. The Geographic Information System (GIS) plugins from Atol CD fill this gap by providing the essential base features required to handle spatial data within Hop's existing infrastructure. Moreover, Hop's ability to execute external processes enables the use of tools such as GDAL - one of the most widely used open-source libraries for geospatial data processing - and its ogr2ogr command-line tool, a powerful vector data converter that supports an additional 100 file formats.

The GIS plugins by Atol CD form a closed system. The seven provided transforms, including the two dedicated to reading and writing files, work with a custom geometry field type based on the Geometry class of the Java Topology Suite (JTS) library. This geometry field type is required for optimised execution of operations. This means that alternative representation formats, such as Well-Known Text (WKT) and Well-Known Binary (WKB), as well as point coordinates split across two fields, must be converted to this geometry type. Without this conversion, external sources such as Excel or CSV files cannot be accessed by the GIS plugins.
Regarding ogr2ogr, it would be beneficial to integrate it more directly into Hop instead of running it externally. This would also resolve the current issue of the GIS file input transform crashing, as it does not wait for preceding transforms to finish.

Two plugins were developed to tackle these problems: Geometry Fields Converter and OGR Vector Import/Export. The Geometry Fields Converter transform allows users to interchangeably convert between WKT, WKB and point coordinate representations. To ensure compatibility with GIS plugins, it is necessary to adjust the field metadata beforehand using the Select values transform. This approach has the advantage of decoupling the plugin from the GIS plugins, enabling it to operate independently as a standalone component. The implementation converts any supported input into a JTS Geometry object prior to converting it into the desired format, using the library's parsers for validation.
The OGR Vector Import/Export plugin provides a transform and an action with a dedicated user interface. This includes a dropdown menu of available vector formats and utilises the locally installed GDAL instance to keep the plugin lightweight. Additional options can be defined manually by advanced users in a tabular input field. In the future, an option to directly convert WKT into a geometry field may be added to the Geometry Fields Converter.

Management Summary

Introduction

Apache Hop is an open-source solution for both enterprise and individual data orchestration provided by the Apache Software Foundation. It allows users to define streamlined networks of transforms to manipulate data on a row-by-row basis, where each field within a row is described by metadata. The wide array of available options and transformation possibilities, which is continuously expanding, establishes Hop as a flexible and extensible platform suitable for a variety of data integration scenarios.

In areas where Hop does not provide native functionality, plugins can be used to extend its capabilities. For example, Apache Hop does not natively support geographic information system (GIS) data. To address this limitation, the French-based company Atol CD developed its own GIS plugins, a set of transforms that provide basic geospatial support. These transforms include both a GIS file reader and writer capable of handling formats such as Drawing Exchange Format (DXF), ESRI Shapefiles, GPS Exchange Format (GPX), GeoJSON, GeoPackage, MapInfo Interchange Format, and SpatiaLite SQLite files. In addition, the plugin suite provides five further transforms offering basic geospatial operations, covering the fundamental requirements for GIS support.

In the context of GIS data, the Geospatial Data Abstraction Library (GDAL) is a highly versatile library that covers a wide range of geospatial processing requirements. One of its components, ogr2ogr, is a powerful command-line tool that enables conversion between numerous GIS file formats, provided the required drivers are installed. Apache Hop provides a transform for executing external processes installed on the user's machine. When working with GIS data in Hop, ogr2ogr can therefore be invoked via the Execute a process transform.

Problem Description

The GIS operation transforms by Atol CD require their input to contain at least one field of the type *Geometry*, a custom metadata field type introduced by the GIS plugins to store geometric values. This requirement significantly restricts the use of other file formats, as the Geometry field type is created only by the GIS file reader or by manually casting fields using the Select values transform. In practice, geometries are often stored in CSV files, Excel files, or other formats supported by Hop, using representations such as Well-Known Text (WKT) or Well-Known Binary (WKB). Point geometries may also be represented by separate coordinate fields. For example, a point with coordinates (5, 3) may be represented as `'POINT (5 3)'` in WKT, as `0101000000000000000000144000000000000008 40` in WKB, or across two separate fields containing `5` and `3`. None of these representations are accepted by the GIS plugins, effectively excluding such datasets from GIS processing within Hop.

Another limitation of the GIS plugins is that the GIS file reader supports only a limited number of formats, while many additional GIS formats exist. As a result, the Execute a process transform is frequently required to convert unsupported formats into one of the seven supported formats.

A more integrated and permanent solution with dedicated configuration options would therefore be highly beneficial.

Technologies

As Apache Hop and all its plugins are written in Java, any plugin solution will also have to be implemented in Java. For geometry support, the Java Topology Suite (JTS) is used; it is also employed in the GIS plugins and provides comprehensive functionality for handling geometries, including readers and writers, factories, and numerous helper methods. Although GDAL will be used only indirectly in the project, a detailed examination of ogr2ogr is necessary to correctly represent its functionality, error codes, supported formats, and behavior in the graphical user interface.

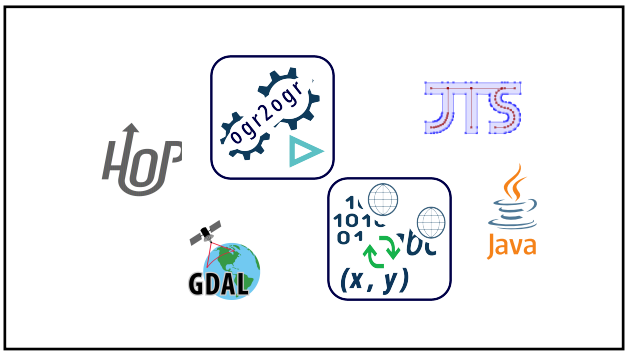

Figure 1: **Important parts of the software stack of the project with the two plugins in the middle** (Apache Hop, GDAL, JTS and Java Icons)

Result

Although both issues address core GIS-related functionality, they were implemented as two separate plugins to keep the solutions focused on one dependency and manageable: the **Geometry Fields Converter** transform and the **OGR Vector Import/Export** transform and action.

The **Geometry Fields Converter** transform, based on the JTS geometry class, allows users to convert geometries interchangeably between WKT, WKB, and point coordinate field formats, while also providing the option to specify the byte order for WKB. Geometries stored in well-known formats may include a Spatial Reference Identifier (SRID), resulting in the expanded formats EWKT and EWKB. The transform implicitly supports these expanded formats and even offers the option to prepend an SRID to well-known geometries.
Figure 2 illustrates a sample use case: an Excel dataset containing points of interest is imported, with coordinates stored in two separate columns, ‘lon’ and ‘lat’. The Geometry Fields Converter transform then combines these coordinates into a single string-type output field named ‘geometry’. Empty values are filtered and the ‘geometry’ field is cast from *String* to *Geometry*, steps required by the GIS file output transform, before the data is exported as a GeoJSON file.

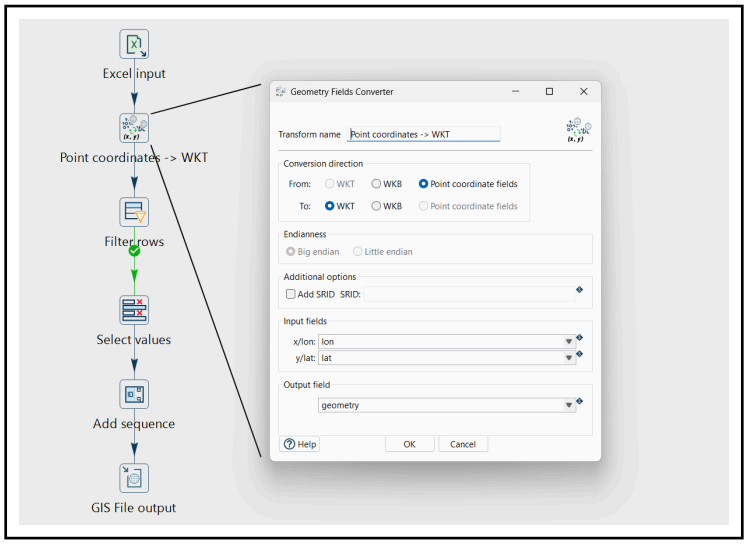

Figure 2: **Hop pipeline using the Geometry Fields Converter transform to convert the point coordinates of an Excel into a GeoJSON file.**

The **OGR Vector Import/Export** plugin provides both an action and a transform solution for ogr2ogr integration. It relies on the user's locally installed instance of GDAL, avoiding unnecessary bloat. Both versions include a tabular field allowing users to specify options at their discretion. Since actions do not produce output data, only the transform version can dynamically read files from field input.
Figure 3 demonstrates the transform converting a file geodatabase into a CSV file for further processing. The resulting CSV file is imported using the CSV file input transform, after which missing addresses are removed, the rows are sorted, and the data is grouped by author.

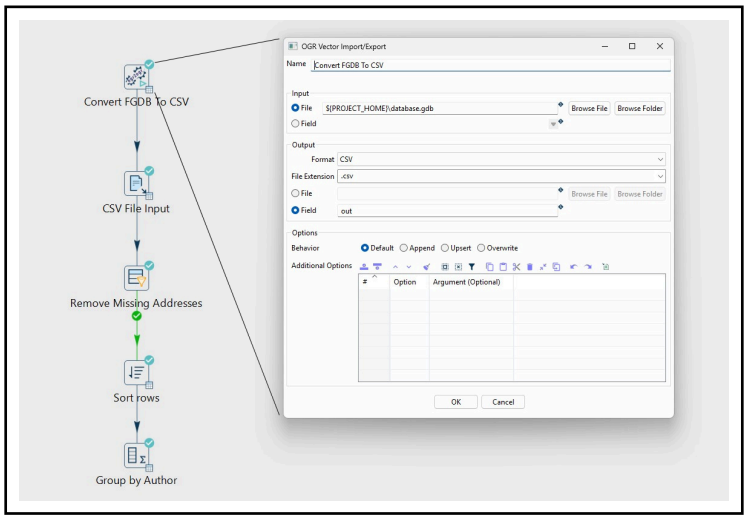

Figure 3: **Using the OGR Vector Import/Export transform to convert a file geodatabase into a CSV file for further processing.**

Outlook

While the main goals have all been been achieved, workarounds are still required in some cases. Logical extensions for the Geometry Fields Converter include adding support for the JSON format and providing the option to save geometries directly as a *Geometry* field type, bypassing the need for the Select values transform. The plugin will be maintained by the Institute for Software after the project's conclusion.

Table of Contents

1 Introduction

This part aims at familiarising the reader with the project by providing an overview of the project's framework conditions within the scope of the semester thesis, the technologies used, and the task itself.

1.1 Motivation

Apache Hop is a platform that enables intuitive and quick implementation of the extract, transform, load (ETL) process. There are several comparable tools available, including Pentaho Data Integration (formerly known as Kettle) and, in the geospatial domain in particular, the Feature Manipulation Engine (FME) and the QGIS Model Designer, a component of QGIS.

Hop takes advantage of being open source and being designed to cover a wide array of data. Through pipelines and workflows, users can define and automate sequences of operations and transform their data.

While many firms rely heavily on geospatial data, Apache Hop does not natively support GIS formats. The company Atol CD provides a solution to this problem with their GIS Plugins (Atol CD, 2025), allowing the import and export of GIS files and adding basic geospatial operations and transformations, covering the most fundamental GIS capabilities.

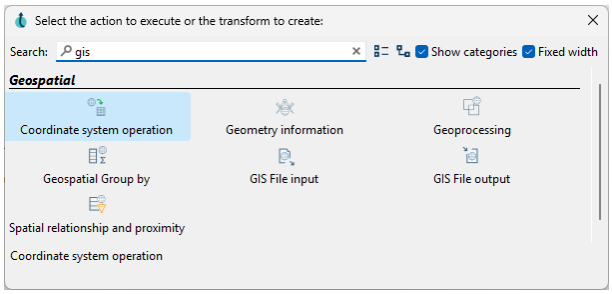

Figure 4: Apache Hop GIS Plugins by Atol CD with all available transforms.

While enabling basic geospatial capabilities, the GIS plugins face several major limitations. The most notable is the limited number of supported GIS file formats; currently, only Drawing eXchange Format (DXF), ESRI shapefiles, GPS eXchange Format (GPX), GeoJSON, GeoPackage, MapInfo Interchange Format (MIF), and SpatiaLite SQLite can be imported.

Another limitation is that all geospatial operation transforms require at least one input field to be of their custom *Geometry* type. Other fields are not selectable, effectively excluding other geometry representation formats such as well-known text (WKT), well-known binary (WKB), and point coordinate fields (see Chapter 1.4.6). Fields are only generated as a *Geometry* type by the GIS reader or manually via the Select values transform, which only works reliably for WKT.

A further limitation is that the GIS File Input transform is always designed to be the first transform in a pipeline. Although it accepts input, it does not use it, and the user must hardcode a file name, preventing dynamic input through fields. Once the pipeline starts, the GIS File Input

immediately attempts to open the specified file. This makes it impossible to import and convert a GIS file within the same pipeline, as the file does not yet exist, causing the pipeline to crash.

For companies working with geospatial data, these capabilities are essential prerequisites for considering Apache Hop as a viable solution.

1.2 Task

The goals of this project are to develop two plugins for Apache Hop in Java, integrate an external process, and provide demos, application examples, and documentation on the Project Hop GitHub (Apache Software Foundation, 2021). Of particular interest is the combination of plugin development in Java for an established project with an established format, parallelisation of an external process, and the handling of productive data pipelines. In addition, a second plugin is to be developed that allows users to convert the representation style of geometries.

1.3 General Conditions

According to the module description (*Modulbeschreibung - Studienarbeit Informatik*, 2023), the thesis should demonstrate problem-solving skills using engineering methods and consist of a conceptual, theoretical and practical part. Successful completion of the thesis earns 8 ECTS credits, which corresponds to a workload of approximately 240 hours per person.

The start date is 15 September 2025 and the submission deadline is 19 December 2025 at 17:00. The exact module description can be found in Appendix Section A.1.

1.4 Technologies

This part delves into all used (and considered) technologies throughout the project, their history and the team's rationale for using them.

1.4.1 Apache Hop

Apache Hop (or the Hop Orchestration Platform) is an open-source data integration platform that allows users to define workflows and pipelines following the ETL (extract, transform, load) process to manipulate their data in various ways, from importing/exporting various file formats and database types to applying standardised formats to a field.

What initially started off as a simple fork of Kettle (Pentaho Data Integration) by Pentaho turned into a fully independent and open-source project with an active and growing online community.

Apache Hop provides a myriad of sample pipelines for every aspect, from showcasing single transforms to complete workflows. Figure 5 illustrates a pipeline that converts data from Excel files into JSON objects. It effectively demonstrates Hop's ETL capabilities, as each step performs a single, simple transformation. The data is first extracted from the Excel files, then transformed by mapping all headers to the data (Lookup Header Label) and converting it into a JSON object (DATA_LINE), allowing the user to load the result, potentially into a MongoDB database.

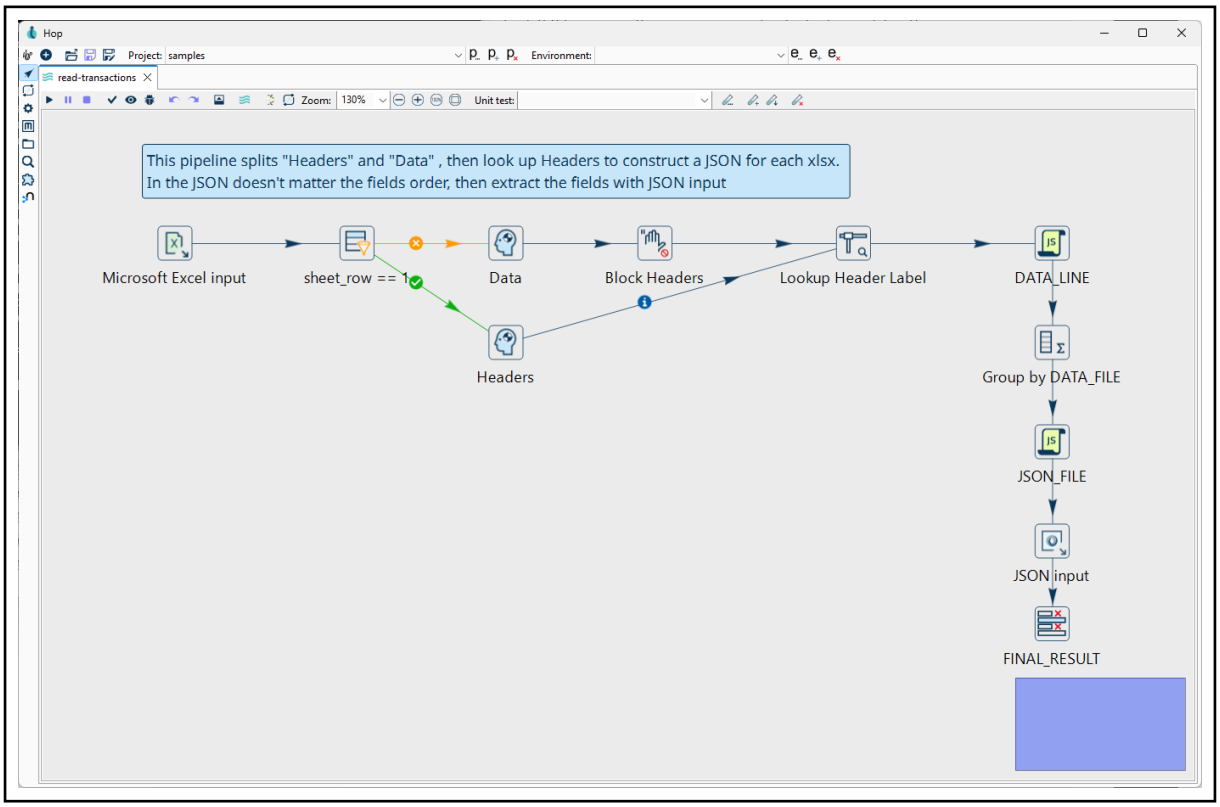

Figure 5: Apache Hop window showing a sample pipeline that constructs a JSON file from Excel sheets.

Also shown in Figure 5 is the branching capability. After the Filter rows transform (*sheet_row == 1*), the two resulting datasets follow separate paths before being merged again at the Lookup step.

Figure 6 effectively demonstrates the levels of complexity that Hop can handle using a very simple workflow. The top-level (parent) workflow calls a sub-workflow, which in turn executes pipelines in parallel (performing a sample task, not relevant here). The dummy actions serve as placeholders, allowing users to experiment with actions.

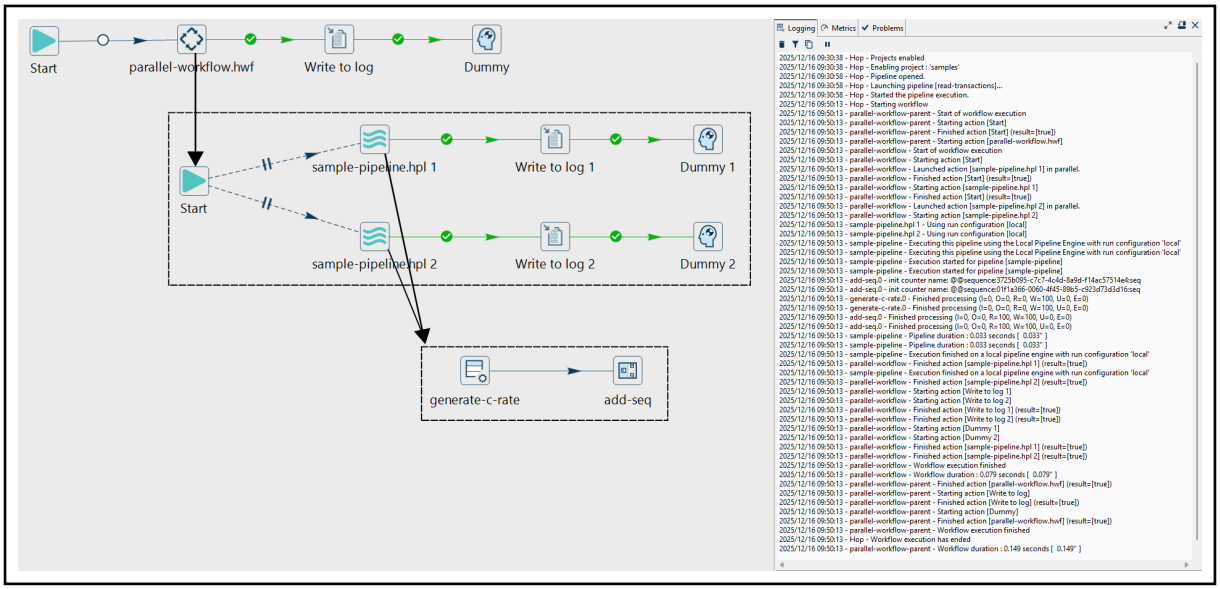

Figure 6: Simple sample workflow showcasing the sequence of actions by logging.

While transforms produce output rows, actions are logical operations that generally do not pass data onwards; instead, they return an error code indicating success or failure. The user can handle failures by branching the workflow according to success or failure outcomes.

In Apache Hop, field types define the kind of data contained in a row, such as numbers, strings, dates, or more complex objects. Each field in a pipeline carries a specific type, which determines how it is processed, validated, and transformed by transforms. Correct field typing is crucial for ensuring compatibility between transforms and for avoiding runtime errors, as many operations rely on the expected data type to function properly. Hop also allows metadata manipulation, enabling users to cast or convert field types when necessary, providing flexibility in data workflows.

1.4.1.1 Atol CD GIS Plugins

Atol CD developed a set of open-source transforms for Apache Hop, adding GIS capabilities such as coordinate system operations, geometry handling, geoprocessing, spatial relationships, GIS file I/O, and geospatial grouping. Maintained under the LGPL-3.0 licence, these transforms extend Hop's core functionality and enable the integration of GIS tasks into data workflows.

The plugin forms a closed system, as every processing transform requires at least one field to be of the plugin's Geometry field type, which is generated either by the plugin's file reader or manually via a field type metadata cast using the Select values transform. This limitation is caused by the input reader not being able to read file names from fields, effectively requiring the user to hardcode file paths directly into the transform.

The problem of the plugin ecosystem being disconnected from Hop's core is further exacerbated by the very limited set of supported formats. The GIS File Input transform supports seven formats: Drawing eXchange Format (DXF), ESRI shapefiles, GPS eXchange Format (GPX),

GeoJSON, GeoPackage, MapInfo Interchange Format (MIF), and SpatiaLite SQLite. The GIS File Output transform additionally supports Keyhole Markup Language (KML) and Scalable Vector Graphics (SVG), but does not support MIF or SpatiaLite SQLite.

Another issue with the GIS file reader is that it is flawed in its current implementation (Prof. Stefan F. Keller, 2025a): it attempts to read the file before the transform is executed, causing pipelines that generate the input file internally to fail. To address the latter limitation, the Institute for Software (IFS) has begun working on a solution. Peter Bühler, a member of the IFS, is currently modifying the GIS File Input transform itself to make it behave consistently with other native Hop file input transforms, such as the CSV File Input. This effort is supported by our team through the provision of sample pipelines and testing.

In parallel, Lars Herrmann, another member of the IFS, developed a workaround. By performing the conversion in a preceding pipeline (see Figure 7), the GIS File Input is able to correctly open the file once the second pipeline (see Figure 8) is started. As Hop operates on a row-by-row basis, a separate instance of the subsequent pipeline is launched for each row. This enables the import of multiple files in parallel, although combining the resulting data from all pipeline instances may prove challenging.

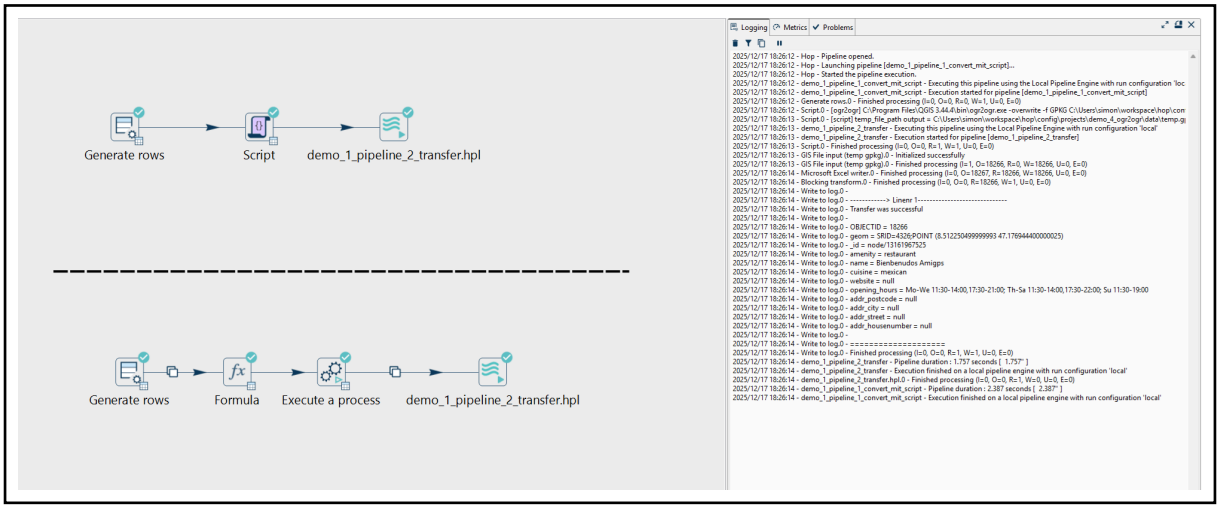

Figure 7: Two pipelines, one using a Script transform and the other the Execute a Process transform, to convert a GIS file before triggering the 'demo_1_pipeline_2_transfer.hpl' pipeline.

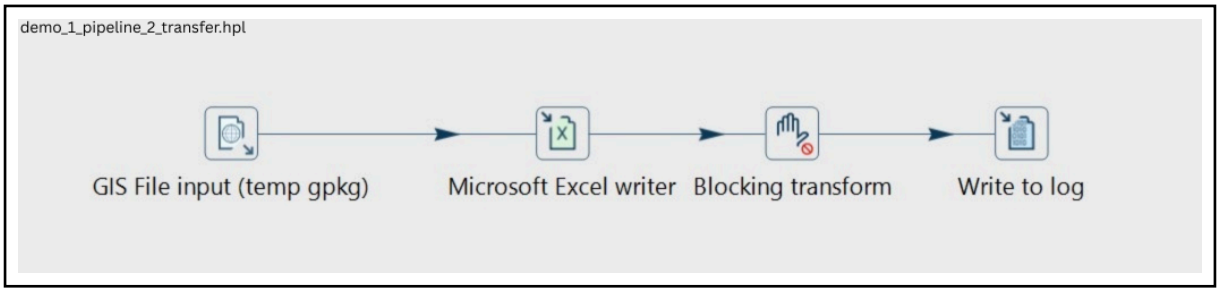

Figure 8: Second workaround pipeline, in which the converted file is imported and saved inside an Excel file.

The Blocking transform and the Write to Log transform are included for debugging purposes, as shown in the Logging tab in Figure 7.

This workaround has been applied to the OGR Vector Import/Export plugin (see Chapter 2.3.2.3).

1.4.2 ogr2ogr

Ogr2ogr is a command-line utility used to convert, reproject, and manipulate vector geospatial data between many different formats (such as Shapefile, GeoJSON, PostGIS, and KML). It is part of the GDAL project, specifically the OGR (vector) component of GDAL, while GDAL itself also handles raster data. ogr2ogr is widely used for tasks like format translation, coordinate system transformation, filtering features, and loading data into spatial databases.

GDAL and its tools, including ogr2ogr, are open-source and were originally developed by Frank Warmerdam and are now maintained by the OSGeo (Open Source Geospatial Foundation) with contributions from a global community.

1.4.3 Java

Java is a general-purpose, object-oriented programming language and platform designed to be portable across systems using the “write once, run anywhere” principle. It is now owned by Oracle and is widely used for building enterprise applications, web services, and a wide range of other applications.

Apache Hop relies on the Long-Term Support (LTS) version of Java Standard Edition 11. Consequently, this project is restricted to the same Java version, which excludes more recent language enhancements, such as:

- Concise switch expressions
- Pattern matching for instanceof
- Text blocks for multi-line strings
- Records and sealed classes
- And other features introduced in Java 12 and later

These limitations do not significantly complicate development but require the team to remain attentive to language constraints.

1.4.4 Java Topology Suite (JTS)

The Java Topology Suite (JTS) is an open-source library providing an implementation of spatial data types and geometric operations in Java. It offers standardised representations for points, lines, and polygons, as well as algorithms for spatial relationships, geometry validation, and topology operations. JTS serves as a foundation for geospatial data processing in Java applications, ensuring correctness and interoperability of geometry manipulations. It is part of the Eclipse Foundation.

Crucial for the Geometry Fields Converter are the library’s parsers, specifically `WKTReader` and `WKBReader`, as well as the writers `WKTWriter` and `WKBWriter` and the `GeometryFactory`. Their use ensures correct and standard-compliant handling of geometries throughout the transform.

1.4.5 Maven

Apache Maven is a build automation and dependency management tool primarily used for Java projects. It standardises how projects are built and how libraries are managed using a declarative POM (Project Object Model) file, and it is an open-source project maintained by the Apache Software Foundation.

Maven provides a variety of plugins that facilitate automation of local deployment, management of dependencies, and quality-of-life enhancements, such as simplified spellchecking.

1.4.6 Geometry Representation Formats

Geometries can be represented in various ways to satisfy different requirements, such as efficient storage, compatibility, or human readability. For comparison, all examples use a point geometry with coordinates x=1.3 and y=3.7.

1.4.6.1 Well-Known Text

Well-Known Text (WKT) represents geometries as strings and is the most common format due to its human readability. It simply states the geometry type along with its coordinates.
WKT: `'POINT(1.3 3.7)'`
WKT can also store an SRID to indicate a specific coordinate system. This is known as Extended Well-Known Text (EWKT).
EWKT: `'SRID=4326;POINT(1.3 3.7)'`

1.4.6.2 Well-Known Binary

Well-Known Binary (WKB) is the most space-efficient format, as geometries are stored directly in binary. To offer at least some human readability, it is often represented in hexadecimal.

The total size depends on the geometry type:
- The first byte specifies the byte order (endianness): `00` for big endian, `01` for little endian.
- The next four bytes indicate the geometry type.
- All subsequent bytes store the coordinates, with each 2D coordinate using 8 bytes for X and 8 bytes for Y.
- Optional SRID information, which specifies the spatial reference system, adds 4 bytes.

Thus, geometries with multiple dimensions or an SRID require more storage than simple 2D geometries, and the total size grows proportionally with the number of points and dimensions.
WKB: `0101000000CDCCCCCCCCCCF43F9A99999999990D40`

#	Bytes	Value
1	`01`	Indicates little endian byte order.
2-5	`01000000`	Represents geometry type = 1 (POINT).
6-13	`CDCCCCCCCCCCF43F`	x-coordinate = 1.3 in little endian double precision.
14-21	`9A99999999990D40`	y-coordinate = 3.7 in little endian double precision.

WKB can also include an SRID, resulting in Extended Well-Known Binary (EWKB). The SRID is represented by setting a flag in the geometry type and including the SRID value directly between the geometry type and coordinates.
EWKB: `0101000020E6100000CDCCCCCCCCCCF43F9A99999999990D40`

#	Bytes	Value
1	`01`	Same as before.
2-5	`01000020`	The last byte of these 4 bytes, `20`, is the SRID flag, the first byte `01` indicates a point geometry type.
6-9	`E6100000`	Represents SRID = 4326 in little endian.
10-17	`CDCCCCCCCCCCF43F`	x-coordinate = 1.3, same as before.
18-25	`9A99999999990D40`	y-coordinate = 3.7, same as before.

1.4.6.3 Point Coordinates

Point coordinates represent geometries by storing each coordinate as a separate numeric value, usually as doubles. This format is extremely efficient for single points, as it avoids the overhead of specifying geometry type or byte order.

Our example would stay the same: x = 1.3 and y = 3.7.

Point Coordinates can optionally include additional dimensions (Z, M) by adding 8 bytes per dimension per coordinate. An SRID can also be stored separately to indicate the spatial reference system. This format is highly compact for points but does not inherently describe geometry type or structure for more complex geometries.

1.4.6.4 JSON

Although not supported in the Geometry Fields Converter, geometries can also be represented using JSON objects, usually inside GeoJSON files, however not necessarily. JSON is highly readable and widely supported, making it convenient for data interchange, web applications, and pipelines.

JSON:

```
{
  "type": "Point",
  "coordinates": [1.3, 3.7]
}
```

JSON can also include additional properties, such as SRID or metadata, although SRID is not part of the official GeoJSON specification. While readable and flexible, JSON is less space-efficient than WKB or Point Coordinates due to the overhead of text formatting and delimiters.

JSON with SRID has no official name.

JSON with SRID:

```
{
  "type": "Point",
  "coordinates": [1.3, 3.7],
  "crs": {
    "type": "name",
    "properties": {
      "name": "EPSG:4326"
    }
  }
}
```

1.4.6.5 Other Formats

There exists a vast number of additional geometry formats, such as Geography Markup Language (GML), FlatGeobuf, and many others, most of which are outside the scope of this project and not relevant.

2 Product Documentation

This part aims at describing the finished project with various visualisations, diagrams and specifications, to provide an understanding of how the final product came to be.

2.1 Requirements Analysis

Requirements lay the foundation of a software product, describing its characteristics and qualities. The closer they align to what the stakeholders have in mind, the better the product. They serve as the cornerstones of quality.

2.1.1 Functional Requirements

Since the plugins involve only one actor, the user, the formulation of functional requirements is rather straightforward.

2.1.2 Epics

Epics cover their main user story and all related tasks required to achieve the desired outcome. One Epic covers the entirety of a plugin.

Jira Link	Epic
AHP-13	Geometry Fields Converter Plugin
AHP-14	OGR Vector Import/Export Plugin

Table 1: Epics.

2.1.3 User Stories

The five user stories define the use cases of the two plugins specifically.

User Story	Jira Epic	Description
Geometry Format Conversion (AHP-34)	(AHP-13)	As a user, I want to convert the representation of geometry (WKT, WKB or Point Coordinates) of my GIS files interchangeably inside a Hop pipeline, so I can use them further in my pipeline.
SRID Support (AHP-60)	(AHP-13)	As a user, I want to keep the SRID of my geometries after conversions and have the option to add one.
Endianness Support (AHP-61)	(AHP-13)	As a user, I want to define the byte order of my WKB-formatted geometry.
OGR Conversion (AHP-36)	(AHP-14)	As a user, I want to convert my OGR format files into different OGR formats files, so I can use them further in my pipeline/workflow.
Additional OGR Options (AHP-64)	(AHP-14)	As a user, I want to be able to use all available ogr2ogr command options in my transform.

Table 2: User Stories.

2.1.4 Non-Functional Requirements

The following subchapter lists all defined non-functional requirements (NFR) according to the ISO/IEC 25010:2023 standard for quality criteria. (*ISO/IEC 25010:2023*, 2023)
Unless explicitly stated otherwise, all requirements apply to both Plugins and must therefore be tested and validated for each. All product modifications are made in consideration of these NFRs; however, the final verification is conducted during the validation tests.
Landing zones are defined for NFRs with quantifiable, numeric performance goals.

2.1.4.1 NFR #1

NFR #1 - Installability	
Category	Flexibility
Description	The installation of the plugin should be easy, straightforward and fast, allowing even inexperienced users to install the plugin without problems.
Goals	• The plugin repository offers an archive that can be dragged and dropped into the Apache Hop Plugin directory. • An installation guide is included in the Readme as well as the OpenSchoolMaps tutorials. • Installation is equal across Windows, Linux and macOS operating systems.
Priority	Very High
Creation date	09.10.2025
Verification dates	**Geometry Fields Converter Plugin:** 12.12.2025
	OGR Vector Import/Export Plugin: 13.12.2025
Remarks	-

Table 3: NFR 1.

2.1.4.2 NFR #2a

NFR #2a - Scalability (Geometry Fields Converter)	
Category	Flexibility
Description	As Geofiles can vary greatly in size, consistent and scalable conversion is essential. As Hop pipelines use parallelisation by default, the parallelisation speed factor is expected to increase the speed per row by a huge margin. For validation, the speed of conversion of a small (1.29 MB) GeoJSON from WKT to WKB is compared to the same conversion of a larger (6.1 MB) GeoJSON. Speed is a pipeline metric in Apache Hop that measures the execution speed in rows per second for a transform. If performance becomes memory-bound, this condition must be explicitly noted in the Readme.
Goal	**Minimal:** Speed ≥ 1.0x **Target:** Speed ≥ 1.5x **Outstanding:** Speed ≥ 2.5x
Priority	Very High
Creation date	11.10.2025
Verification date	12.12.2025
Remarks	Outstanding goal achieved, with a final ratio of 3.4x.

Table 4: NFR 2a.

2.1.4.3 NFR #2b

NFR #2b - Scalability (OGR Vector Import/Export transform)	
Category	Flexibility
Description	As Geofiles can vary greatly in size, consistent and scalable conversion is essential. Streams and/or parallelisation shall be utilised to make this possible. For validation, the speed for a conversion of a small (1.29 MB) GeoJSON to a GeoPackage is compared to the conversion of a larger GeoJSON file (6.1 MB). If performance becomes memory-bound, this condition must be explicitly noted in the Readme.
Goal	**Minimal:** Speed ≥ 0.21x (1.29 MB / 6.1 MB) **Target:** Speed ≥ 0.4x **Outstanding:** Speed ≥ 0.75x
Priority	Very High
Creation date	11.10.2025
Verification date	13.12.2025
Remarks	Minimal goal achieved, with a final ratio of 0.31x.

Table 5: NFR 2b.

2.1.4.4 NFR #2c

NFR #2c - Scalability (OGR Vector Import/Export action)	
Category	Flexibility
Description	As Geofiles can vary greatly in size, consistent and scalable conversion is essential. Streams and/or parallelisation shall be utilised to make this possible. For validation, the speed for a conversion of a small (1.29 MB) GeoJSON to a GeoPackage is compared to the conversion of a larger GeoJSON file (6.1 MB). If performance becomes memory-bound, this condition must be explicitly noted in the Readme.
Goal	**Minimal:** Speed ≥ 0.21x (1.29 MB / 6.1 MB) **Target:** Speed ≥ 0.4x **Outstanding:** Speed ≥ 0.75x
Priority	Very High
Creation date	11.10.2025
Verification date	13.12.2025
Remarks	Minimal goal achieved, with a final ratio of 0.33x.

Table 6: NFR 2c.

2.1.4.5 NFR #3

NFR #3 - Adaptability	
Category	Flexibility
Description	As Hop is available on all major platforms (Windows, Linux, macOS), the plugin and all its functionalities should be platform-independent.
Goal	The plugin and all its use cases are platform-independent.
Priority	High
Creation date	11.10.2025
Verification dates	**Geometry Fields Converter Plugin:** 12.12.2025
	OGR Vector Import/Export Plugin: 13.12.2025
Remarks	-

Table 7: NFR 3.

2.1.4.6 NFR #4

NFR #4 - Open Source	
Category	License Compliance (Non-standard)
Description	To support future Apache Hop plugin developers, the plugin's source code is provided as open-source software under the Apache License (*ASF Source Header and Copyright Notice Policy*, 2004).
Goals	• The plugin accurately implements the Apache License 2.0 (*ASF Source Header and Copyright Notice Policy*, 2004). • The source code's repository is set to public. • A comprehensive Readme of the software architecture is located inside the public Git repository.
Priority	High
Creation date	15.10.2025
Verification dates	**Geometry Fields Converter Plugin:** 12.12.2025
	OGR Vector Import/Export Plugin: 13.12.2025
Remarks	-

Table 8: NFR 4.

2.1.4.7 NFR #5

NFR #5 - Interoperability	
Category	Compatibility
Description	The plugin is fully integrated into Apache Hop, allowing users to interact with other transforms or externally with other programs after an export write.
Goals	• Users are able to apply the plugin's transform within a pipeline to manipulate previous outputs and pass the resulting data to subsequent steps in the pipeline. • After an export, the exported file is compatible with and can be processed by other programs.
Priority	High
Creation date	15.10.2025
Verification dates	**Geometry Fields Converter Plugin:** 12.12.2025
	OGR Vector Import/Export Plugin: 13.12.2025
Remarks	-

Table 9: NFR 5.

2.1.4.8 NFR #6

NFR #6 - Operability/Self-Descriptiveness	
Category	Interaction Capability
Description	The plugin follows the general aesthetic, UI structure, and look and feel of existing plugins, enabling users already familiar with Apache Hop to work with it seamlessly.
Goals	The plugin interaction window is built similarly to pre-existing plugins.
Priority	Medium
Creation date	15.10.2025
Verification dates	**Geometry Fields Converter Plugin:** 12.12.2025
	OGR Vector Import/Export Plugin: 13.12.2025
Remarks	-

Table 10: NFR 6.

2.1.4.9 NFR #7

NFR #7 - Functional Appropriateness (OGR Vector Import/Export)	
Category	Functional Suitability
Description	The Atol CD GIS plugins already implicitly provide the ability to convert vector files, by importing the file into Hop and then exporting it in the targeted format. As the number of supported files is limited to seven (.dxf, .shp, .gpx, .geojson, .gpkg, .mif and .sqlite), the goal is to massively expand on the number of supported formats to make full use of the GDAL/OGR library.
Goals	**Minimal:** The plugin is able to convert between (.dxf, .shp, .gpx, .geojson, .gpkg, .mif and .sqlite) files and one additional file. **Target:** The plugin is able to convert at least half of the supported OGR formats (see (*Vector Drivers*, 2025)). **Outstanding:** The plugin is able to convert all of the supported OGR formats.
Priority	High
Creation date	15.10.2025
Verification date	13.12.2025
Remarks	Outstanding goal met, the Plugin supports all OGR formats, however only as long as all drivers are installed on the user's system.

Table 11: NFR 7.

2.2 Architecture

This chapter explores the architectural aspects of Apache Hop plugin development, focusing on the structure and components needed to create custom transforms and actions. It provides insights into the build process, plugin installation, and the core classes that make up a typical plugin. Since the final product is a plugin within an existing framework, it is difficult to represent the plugins themselves effectively with diagrams or graphs. Therefore, this chapter emphasises explaining Apache Hop's architecture and the plugin development process rather than visual representations of the plugins.

2.2.1 Hop Architecture

Apache Hop describes itself as entirely metadata driven, with every object type defining how data is processed and how pipelines and workflows are orchestrated. Internally, it employs a kernel architecture, and plugins extend functionality through their own metadata (*Apache Hop*, 2025).

Figure 9 showcases the kernel architecture.

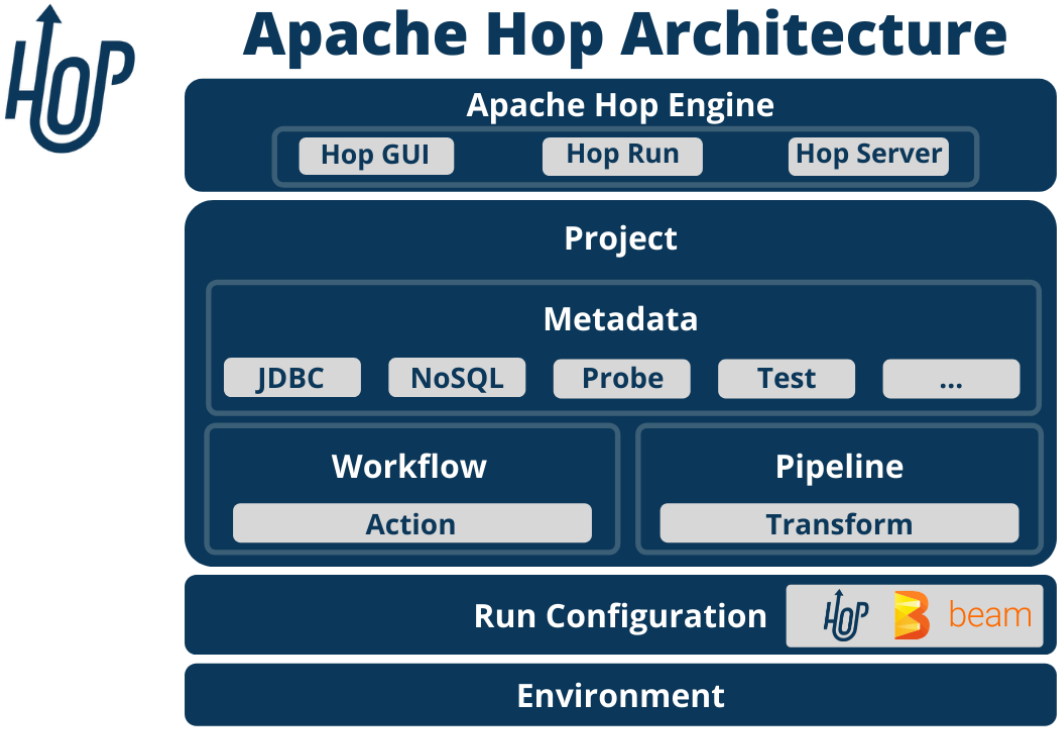

Figure 9: Apache Hop architecture diagram (*Apache Hop Architecture*, n.d.).

The **Environment** (short for Project Lifecycle Environment) provides environment-specific variable definitions across projects, typically set up according to development phases such as Development, Test, Acceptance, and Production (*Projects & Environments*, 2025).

A **Project** is a collection of all files required for a data orchestration solution, typically including metadata, pipelines, workflows, reference files, and documentation (*Projects & Environments*, 2025).

The **Run Configuration** allows users to define how a pipeline or workflow should run, specifying the engine, parameters, and variables. For example, in Figure 10, the `${PROJECT_HOME}` project variable, which defines the project location, is configured.

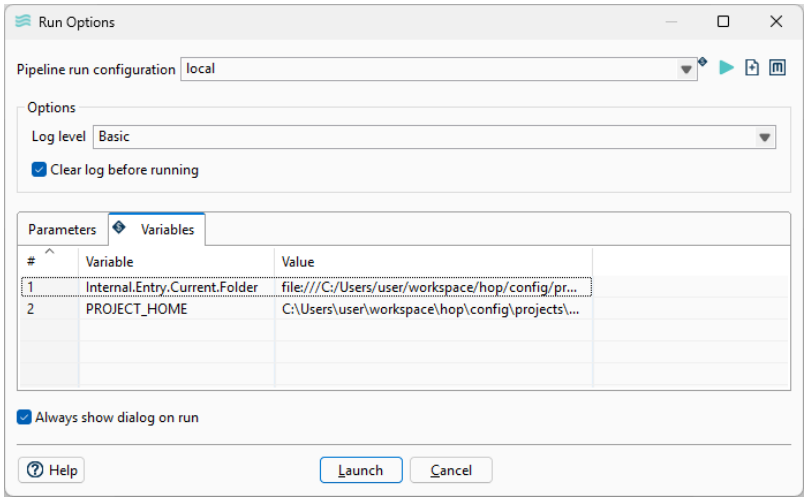

Figure 10: Standard pipeline run configuration.

Metadata are the basis to everything in Hop and are saved as JSON files.

The **Engine** is the core component responsible for executing pipelines and workflows. It interprets metadata to read, transform, and write data, integrates plugin functionality, and operates within the context of a project, environment, and run configuration. Users can access the engine via three tools: **Hop GUI**, offering a full graphical interface; **Hop Run**, a CLI version without a UI; and **Hop Server**, which provides a lightweight server for REST calls.

2.2.2 Plugin Development

Due to the limited availability of official documentation, developing Apache Hop plugins from scratch can be challenging at first. This chapter describes the general architecture of Apache Hop transforms and actions, outlines the required development environment, and explains the implementation of said plugin types.

Apache Hop provides an official plugin skeleton repository (Apache Software Foundation, 2022), which serves as a reference implementation and starting point for custom plugin development. The plugins discussed in this chapter are based on these samples, in particular the hop-transform-sample and hop-action-sample projects.

2.2.2.1 Build process and plugin installation

Apache Hop plugins are distributed as Java archive (.jar) files. Installation is performed by copying the compiled jar into the plugins directory of the Hop installation and placing it into the subdirectory corresponding to the plugin type, for example:

```
plugins/transforms/transformName
plugins/actions/actionName
```

For dependency and build management, Apache Maven is used. Maven is the predominant build tool among existing Hop plugins and integrates well with the Hop plugin skeleton project. Executing `mvn package` produces a deployable .jar file.

The project's Project Object Model (pom.xml) contains custom packaging and dependency configurations for Maven. Using one can create an automatic job that copies the built .jar

file directly into its desired location inside the Hop installation, saving time when working iteratively.

Localisation resources (e.g., labels, tooltips, and descriptions) are stored in the *resources* directory and are resolved at runtime based on the user's locale. Beware that it is important that the package is named correctly.

2.2.2.2 Transform architecture

Apache Hop transforms are used within pipelines to process rows of data. A transform implementation typically consists of four core Java classes, each with a distinct responsibility. The following sections will give an overview of what each class does and how it interacts with Apache Hop.

2.2.2.2.1 Transform execution class

The main execution logic is implemented in a class extending BaseTransform. This class overrides the processRow() method, which is invoked for each incoming row.

```
public class ExampleTransform extends BaseTransform {

  @Override
  public boolean processRow() throws HopException {
    Object[] row = getRow();
    if (row == null) {
      setOutputDone();
      return false;
    }

    // Business logic
    putRow(getRowMeta(), row);
    return true;
  }
}
```

As Hop transforms are running asynchronously, `getRow()` is blocking and will wait until:

a) the previous transform calls `putRow(...)`

b) the previous transform calls `setOutputDone()`, in this case, `row` would be `null`

In case there is no previous transform, like in a file input, `processRow()` will be called once and `row` will be `null`.

Assuming it's not `null`, the `getRow()` method always returns an Array of generic objects that have to be cast manually to the desired type if they are required for an opteration. This can look something like this:

```
IRowMeta inputRowMeta = getInputRowMeta();
int inputFieldIndex = inputRowMeta.indexOfValue('myColumnName');
if (inputFieldIndex < 0) {
  throw new HopException("Field 'myColumnName' not found in input row meta!");
}
String value = (String) row[inputFieldIndex];
```

The parent class `BaseTransform` implements `getInputRowMeta()`.

2.2.2.2.2 TransformMeta class

The metadata class extends BaseTransformMeta and defines the transform's metadata and configuration. User-configurable properties are annotated with `@HopMetadataProperty`, which enables automatic serialisation and persistence. If a custom datatype should be persisted as well, make sure to annotate its properties as `@HopMetadataProperty` as well.

```
public class ExampleTransformMeta extends BaseTransformMeta {

  @HopMetadataProperty(key = "exampleField")
  private String exampleField;
}
```

This class also implements methods that describe the transform's behaviour to the Hop engine, such as:

`getFields(...)` , which defines the structure of the output row metadata as a RowMeta object.

A RowMeta describes the structure of a data row in Apache Hop, including the names, data types, lengths, and formats of all fields. During pipeline execution, RowMetas are propagated between transforms and modified as needed to reflect schema changes, ensuring that both data values and their metadata remain consistent throughout the pipeline.

`getTransformIOMeta()` , which specifies input and output capabilities of said transform.

2.2.2.2.3 TransformDialog class

The configuration dialog is implemented by extending BaseTransformDialog. The dialog defines the graphical user interface used to configure the transform and is implemented using the Eclipse Standard Widget Toolkit (SWT) (Eclipse Foundation, 2025).

```
public class ExampleTransformDialog extends BaseTransformDialog {
  private Text wExampleField;

  public String open() {
    // SWT UI creation
    return transformName;
  }

  private void getData() {
    // load data from meta object
    wExampleField.setText(input.getExampleField());
  }

  private void setData() {
    // save data to meta object
    input.setExampleField(wExampleField.getText());
  }

  @Override
  protected void ok() {
    setData();
    dispose();
  }

  @Override
  protected void cancel() {
    dispose();
```

```
  }
}
```

2.2.2.2.4 TransformData class

The data class extends BaseTransformData and stores runtime data shared between multiple copies of the same transform instance. This is primarily relevant for parallel execution and may be omitted for simpler transforms.

```
public class ExampleTransformData extends BaseTransformData {
  // Runtime data fields
}
```

2.2.2.3 Action architecture

Apache Hop actions are used within workflows to perform sequential tasks, such as file operations or job orchestration. Compared to transforms, actions are generally simpler, as they do not process row data.

An action implementation typically consists of the following components.

2.2.2.3.1 Action execution class

The core logic of an action is implemented by extending the action class. Execution behaviour is defined in the execute() method, which returns a result indicating success or failure.

```
public class ExampleAction extends Action {

  @Override
  public Result execute(Result previousResult, int nr) {
    Result result = previousResult;
    // Action logic
    result.setResult(true);
    return result;
  }
}
```

Action configuration is stored directly within the action class. As with transforms, properties annotated with `@HopMetadataProperty` are automatically serialised and persisted.

```
@HopMetadataProperty(key = "exampleField")
private String exampleField;
```

2.2.2.3.2 ActionDialog class

The graphical configuration interface for an action is implemented by extending ActionDialog. Like transform dialogs, action dialogs are built using Eclipse SWT.

```
public class ExampleActionDialog extends ActionDialog {
  // Similar logic as TransformDialog
}
```

Unlike transforms, actions do not require a separate data or meta class, as they are not executed in parallel and do not operate on row-based data streams.

2.2.3 Architectural Decision Records

During the project several architectural decisions, so-called ‘big decisions’, had to be made. This subchapter is a documentation of every single such architectural decision made at the

Most Responsible Moment (MRM) and recorded in the Y-statement style (Olaf Zimmermann, 2020).

ADR 1	
Date	15.10.2025
Context	Repository
Facing	The need to define the structure of our repositories
We decided	To split our two plugins into two separate repositories
And neglected	Having one monorepo for both plugins
To achieve	Easier CI/CD and being able to offer both plugins as single entities
Accepting	Duplicate code and minor redundancy.

Table 12: ADR 001.

ADR 2	
Date	17.10.2025
Context	Unit Testing
Facing	The need to define the scope of unit testing for our plugins
We decided	To implement unit tests only for the Geometry Fields Converter transform execution class
And neglected	Testing other classes via mocks
To achieve	Keeping testing within the project's scope
Accepting that	The team must manually verify changes in classes like Dialog to prevent regressions.

Table 13: ADR 002.

ADR 3	
Date	24.10.2025
Context	CSV Validator Plugin
Facing	The need to rethink the CSV Validator plugin definition based on new findings during Sprint 3 (see Chapter 3.3.1)
We decided	To keep User Story (AHP-34) unchanged
And neglected	Using Hop static schema mapping combined with turning the CSV Validator transform into a generic input validator
To achieve	Support for all additional CSV Schema Language (*CSV Schema Language 1.2*, 2024) features, such as regular expressions, logical expressions, and elaborate data type enforcement
Accepting that	Validation of other file formats will not be included within the scope of the plugin.

Table 14: ADR 003.

ADR 4	
Date	7.11.2025
Context	CSV Validator Plugin
Facing	The need to rethink the CSV Validator plugin use case based on new findings during Sprint 4 (see Chapter 3.3.2)
We decided	To implement the geometry representation converter
And neglected	Continuing working on the CSV Validator
To achieve	Compliance with the workload requirements set by the module description
Accepting that	All work previously done on the CSV Validator will have to be redone for the new plugin.

Table 15: ADR 004.

ADR 5	
Date	22.11.2025
Context	OGR Vector Import/Export Plugin
Facing	The need to rethink the repository architecture of the plugin
We decided	To keep both action and transform in the same repository and add a submodule for shared code
And neglected	Splitting the two into their own repositories
To achieve	Less duplicate code
Accepting that	The CI/CD pipeline as well as the Maven module architecture will have to be reworked.

Table 16: ADR 005.

ADR 6	
Date	9.12.2025
Context	Geometry Fields Converter Plugin
Facing	The need to decide the degree of coupling to the Atol CD GIS plugins for compatibility
We decided	To establish compatibility via the Select values transform
And neglected	Using the Geometry ValueMeta from Atol CD
To achieve	Decouplement and independency
Accepting that	To be able to use the GIS plugins transforms on the output of the Geometry Fields Converter, users need to add a Select values transform that changes the metadata of the WKT field to a geometry type before the GIS transform.

Table 17: ADR 006.

2.3 OGR Vector Import/Export

As outlined previously in this document, the OGR Vector Import/Export plugins allow users to utilise the ogr2ogr command line utility from within Hop pipelines and workflows. The primary use case of which is to convert GIS files into formats supported by the Atol CD GIS File input.

2.3.1 User Interface

As with all of Apache Hop, the user interface is implemented with the Standard Widget Toolkit (SWT). The action uses a stripped-down version of the transform's dialog. Options related to field-based input and output are not available, as workflows do not operate on row-based data. Furthermore, the 'File Extension' input was also omitted for the action, as it is only necessary for writing temporary files when using 'Field' output.

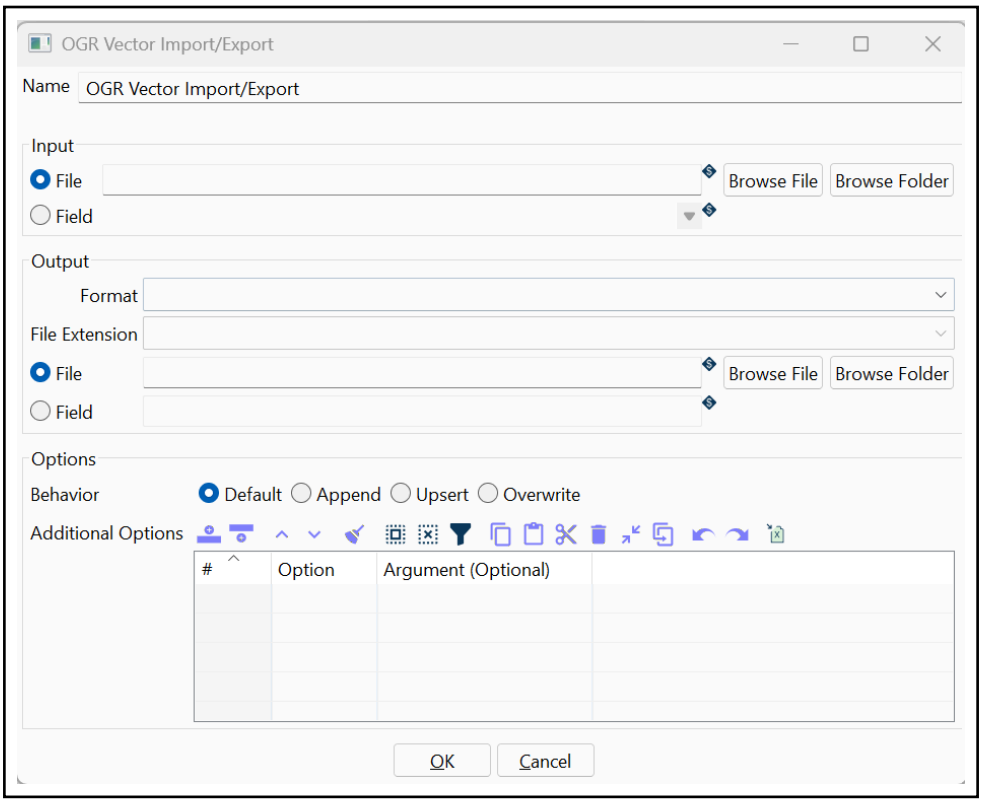

Figure 11: Screenshot of the OGR Vector Import/Export transform dialog.

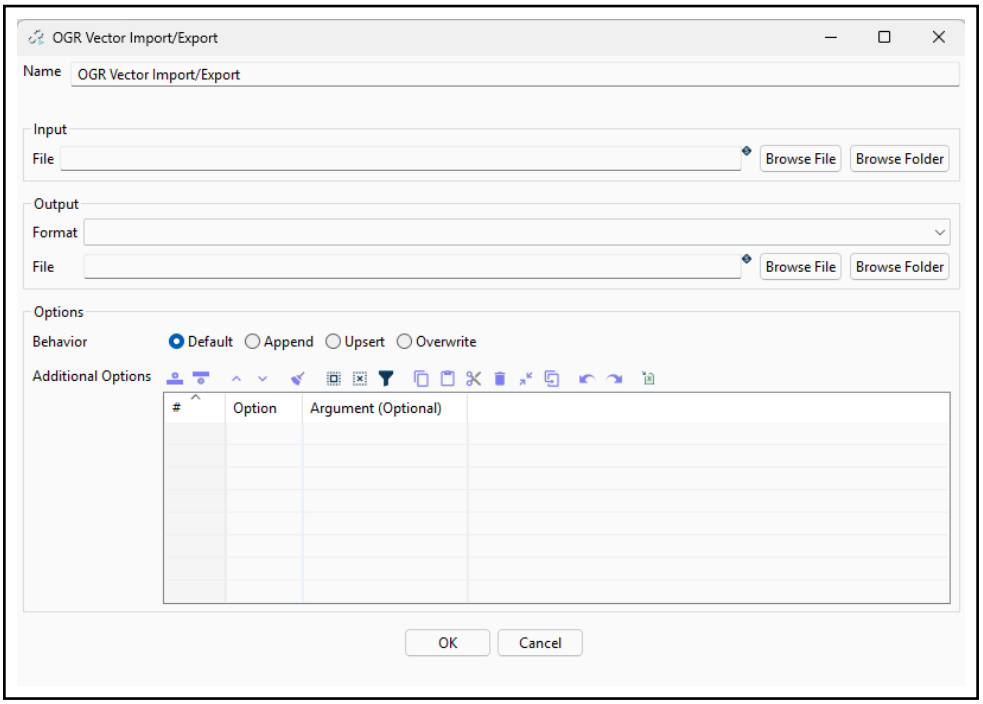

Figure 12: Screenshot of the OGR Vector Import/Export action dialog.

The Additional Options section was implemented as a table rather than a single textbox. This avoids the need to parse and split the options manually, since Java's ProcessBuilder requires arguments to be passed separately, and it also provides a clearer overview of the configured options.

Logo

The logo (see Figure 13) is the same for both action and transform. It is based on the Execute a process transform icon, with the name 'ogr2ogr' included to clearly communicate the core functionality of the plugins.

Figure 13: Logo of the OGR Vector Import/Export.

2.3.2 Usage

This section describes how to configure and use the OGR Vector Import/Export transform and action.

2.3.2.1 Input and Output Types

The transform supports two different input and output modes: File and Field. The selected combination directly influences the runtime behaviour.

Input Types

- File

  A single, explicitly specified input file is used. The file is selected via a file chooser in the transform configuration dialog.

- Field

  The transform reads the path to the input file from a field in the incoming data stream produced by a preceding transform.

Output Types

- File

  The transform writes its result to a single output file.No file path is written to the pipeline.

- Field

  The transform writes the absolute path of the generated output file into a specified field of the outgoing data stream.

When Field output is selected, generated files are stored inside the project's home directory in an automatically created subfolder called ogr_temp. This requires the PROJECT_HOME variable to be set, which is the default behaviour.

Because the plugin cannot determine how long these files are needed, it cannot delete them automatically. To prevent the accumulation of unnecessary files, the Delete folder action can be used to clean up this directory.

Input / Output	File	Field
File	Performs a single OGR conversion using the configured input file. The pipeline continues, but no file path is written to the output.	Performs a single OGR conversion and writes the absolute path of the automatically generated output file to the specified output field.
Field	Reads only the first incoming row and performs a single OGR conversion. The pipeline continues, but no file path is written to the output.	Performs one OGR conversion per incoming row and writes the absolute paths of all generated output files to the specified output field.

Table 18: Runtime behaviour of the OGR Vector Import/Export transform depending on input and output type.

Format

The format specifies the output driver used by `ogr2ogr`. The dropdown menu lists only those formats that are supported by the locally installed `ogr2ogr` instance. If a required format is not available, additional GDAL drivers may need to be installed.

File Extension

The file extension is needed if field output is selected, as some formats support multiple different file extensions. It automatically displays the extensions that `ogr2ogr` lists as standard for the selected format, but any text can be entered if another extension is needed.

Behavior

These radio buttons control how `ogr2ogr` behaves when writing to a file that already exists (e.g., overwrite, append, or upsert). The plugin will fail if the output file already exists and 'Default' is selected.

Additional Options

Any number of additional command-line options can be entered in this table. See `ogr2ogr --help` for further information.

2.3.2.2 Installation

1. Install GDAL, which includes ogr2ogr. If you have QGis installed, GDAL is probably already installed. On Windows we recommend using OSGeo4W to install as it includes a broader set of drivers than other available binaries. Alternatively, GDAL can be built individually.
2. Check with `ogr2ogr --formats` if the drivers you require are installed.
3. Download the latest release from GitLab

or

3. Clone and build it yourself using the appropriate `mvn package` command.
4. Move the extracted folder to your installation folder as detailed below.

For the transform:

```
hop
└── plugins
    └── transforms
        └── ogr2ogr
            ├── hop-transform-ogr2ogr-version.jar
            └── version.xml
```

or for the action:

```
hop
└── plugins
    └── actions
        └── ogr2ogr
            ├── hop-action-ogr2ogr-version.jar
            └── version.xml
```

5. Restart Hop

2.3.2.3 GIS File Input Workaround

As explained in Chapter 1.4.1.1, the GIS File Input transform attempts to read the input file as soon as the pipeline is started, making its combined use with the OGR Vector Import/Export transform impossible. To address this, Lars Herrmann devised a temporary solution.

The workaround was adapted to the plugin and now consists of an OGR Vector Import/Export action within a workflow, after which a pipeline is triggered in which the converted file is imported (see Figure 14). However, this approach requires hardcoding the file name and forfeits the parallel pipeline execution, as actions do not have inputs or outputs. As a result, the number of files that can be converted is limited to one per action.

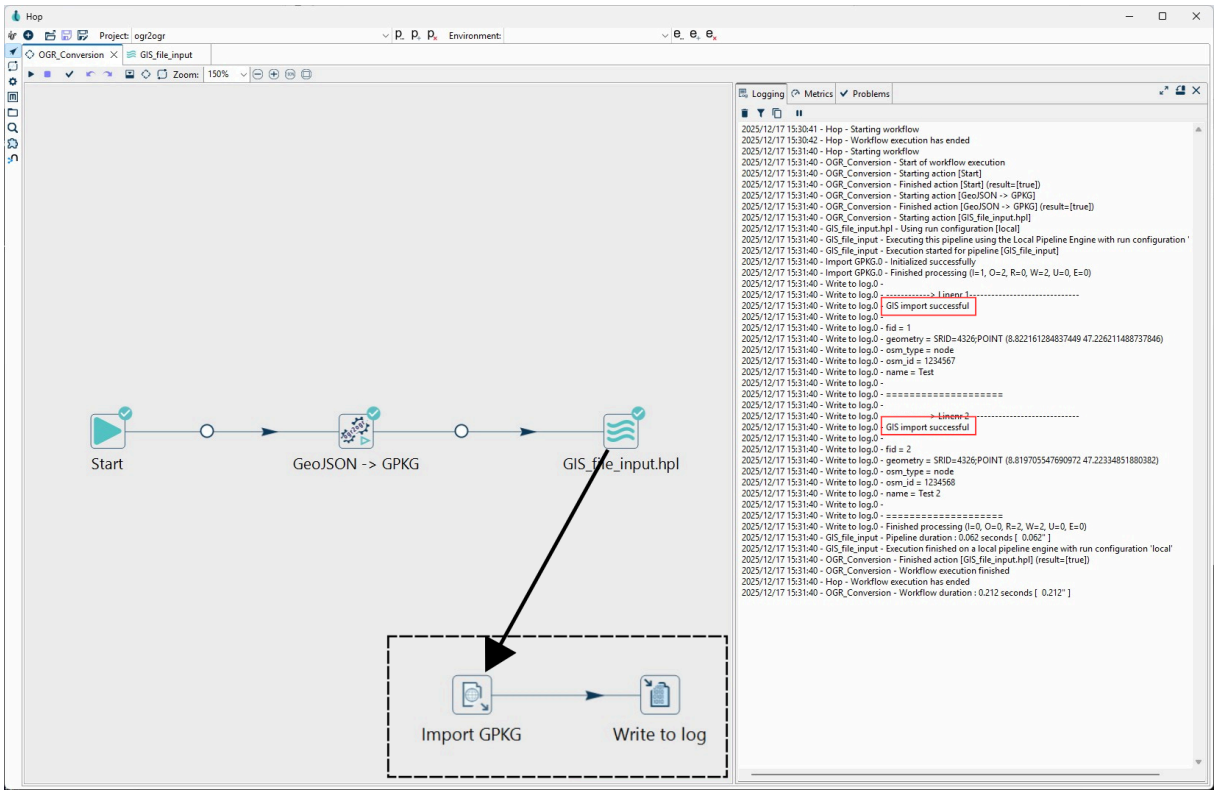

Figure 14: Workflow illustrating the GIS File Input workaround using an action-based conversion followed by a pipeline.

In Figure 14, the log displays the message “GIS import successful,” which is only emitted if the preceding transform, the GIS File Input, completes successfully, thereby demonstrating the effectiveness of the workaround.

2.3.2.4 Examples

Transform

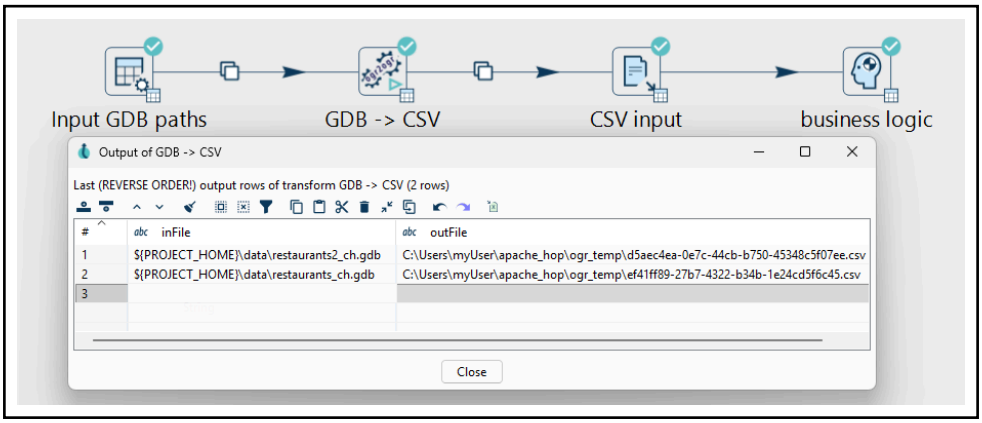

Figure 15: Example of using the transform in a basic pipeline.

This pipeline shows a basic example of how to use the transform to convert multiple files that are not suported by Hop (in this case GDB) into a supported format like CSV. CSV was used in this case because the standard Hop CSV Input supports field based input of file paths. The current official version of Atol CD’s GIS File input does not support that, as previously mentioned.

Action

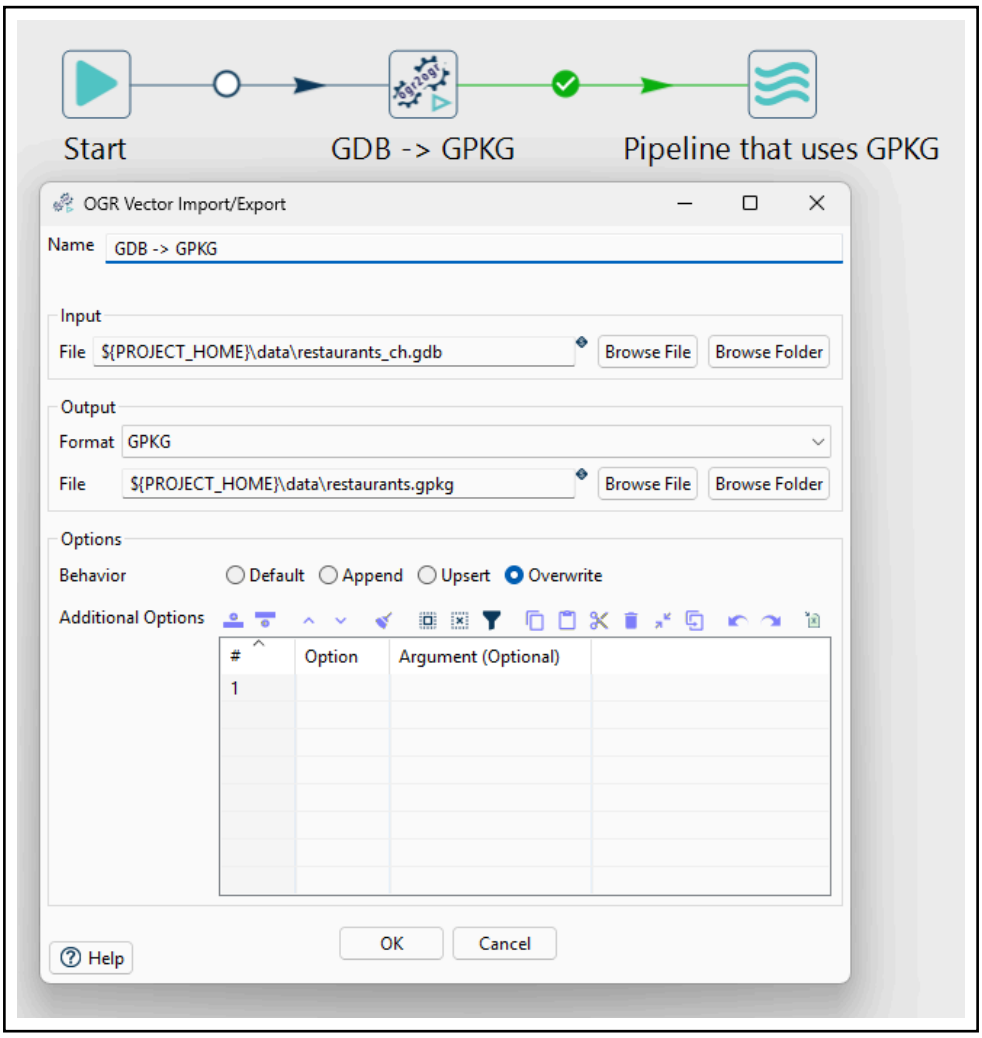

Figure 16: Example of using the action in a basic Workflow.

Using the action can be useful as it doesn't rely on the behaviour of any file inputs. This example converts the GDB to a GPKG file and then calls a pipeline. As the green checkmark on the Hop indicates, the pipeline will only be called if the conversion was successfull.

2.3.3 Implementation

This section describes internal implementation details of the plugin.

2.3.3.1 Process Execution

The plugin uses Java's standard ProcessBuilder API to execute ogr2ogr as an external process.

```
ProcessBuilder pb = new ProcessBuilder(
  "ogr2ogr",
  "-f",
  meta.getFormat(),
  outputPath,
  inputPath
);
```

The standard output and error streams of the process are captured. In case of an error, the output is printed to the Hop console.

2.3.3.2 File and Field Inputs

File input textboxes in the configuration dialogs are implemented as standard text inputs.

These textboxes intentionally accept arbitrary text and are not validated. This allows users to specify non-file-based input and output targets, such as supported databases (e.g., PostgreSQL/PostGIS), assuming the appropriate GDAL drivers are installed.

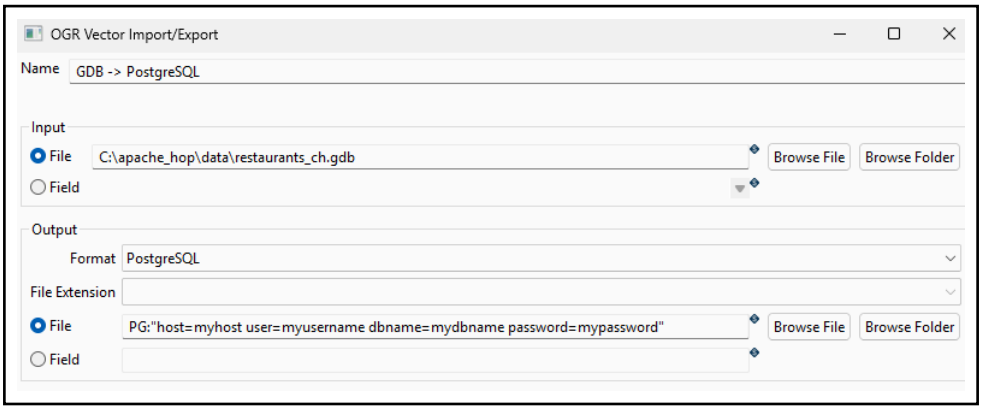

Figure 17: Example of using the transform to write to a PostgreSQL and PostGIS database.

This is not considered an official feature of the plugins and has not been extensively tested. However, we intentionally avoid imposing artificial limitations on the plugins' capabilities.

2.3.3.3 Error handling

The plugin may fail at runtime for several reasons:

- `ogr2ogr` returns a non-zero exit code, indicating an error during execution.
- An exception occurs while invoking or running the `ogr2ogr` command.
- When using 'Field' output, the `ogr_temp` directory cannot be created inside the project home folder.

`ogr2ogr` error messages can be verbose and difficult to interpret. Parsing these messages was not feasible, as their structure varies between GDAL versions and installations.

As a result, error output is forwarded directly without additional interpretation.

For the transform, a Hop internal exception (`HopTransformException`) is thrown. This causes the pipeline to stop and the error message to be written to the 'Logging' section.

For the action, the result object is set to `false`, indicating that the action did not complete successfully. The error is also written to the 'Logging' tab.

Both plugins will appear red with an exclamation mark in the corresponding overview.

2.4 Geometry Fields Converter

The Geometry Fields Converter is the missing link to enable the processing of geospatial data in Apache Hop, without necessitating a GIS file.

2.4.1 User Interface

Like all transforms, the user interface is implemented with the Standard Widget Toolkit (SWT). Figure 18 shows the initial mockup for the dialog, designed to satisfy the non-functional requirement for operability and self-descriptiveness (see Chapter 2.1.4.8) and align the dialog window's look and feel with those of existing transforms.

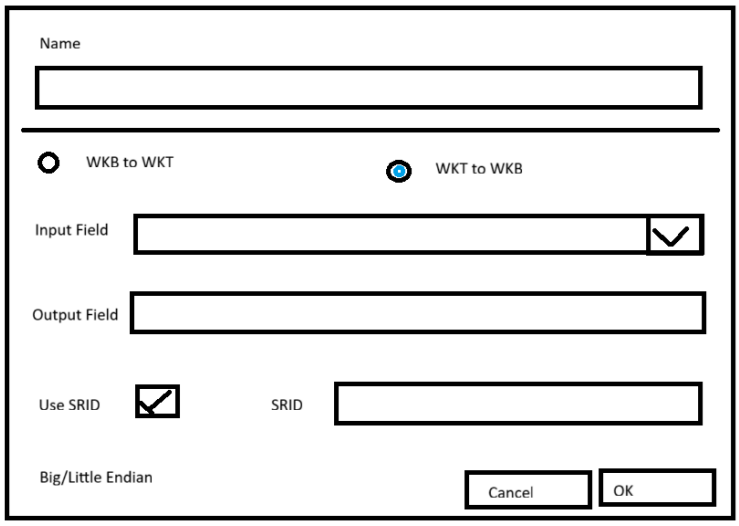

Figure 18: Mockup of the Geometry Fields Converter.

As the transform evolved, with the addition of the point coordinates format, the GUI had to evolve as well, resulting in the split of the conversion direction into output and input format selections. This change allows the plugin to be easier expanded in the future, in case more representation formats are to be supported.

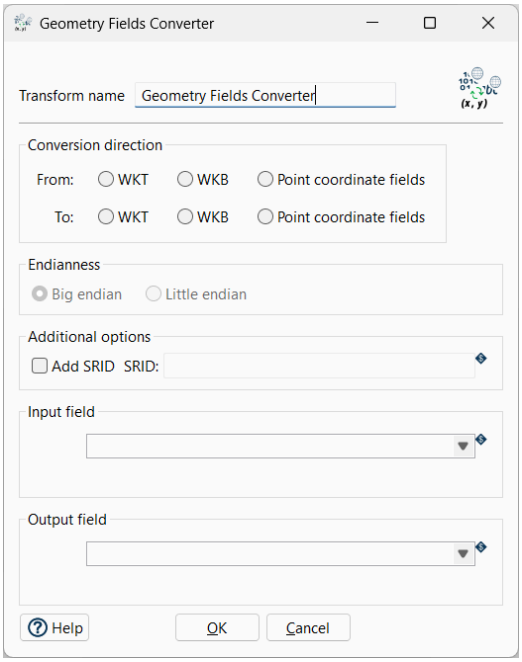

Figure 19: Final version of the GUI of the Geometry Fields Converter.

Logo

The logo (see Figure 20) is an SVG file designed in Inkscape, an open-source SVG editor. It incorporates existing elements of Hop field type graphics for string and binary, the globe icon from Atol CD's GIS plugins transforms, and a custom representation for point coordinates.

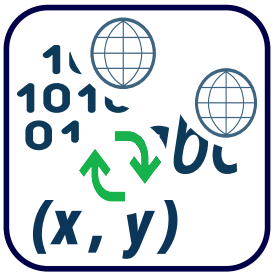

Figure 20: Logo of the Geometry Fields Converter.

2.4.2 Usage

The Geometry Fields Converter follows the standard Hop behaviour for input error handling. Users are responsible for ensuring the correct formatting of input data; incorrect formatting will cause the pipeline to crash as soon as the erroneous row is processed. Apache Hop provides various means to enforce data standardisation, such as the Data validator transform or static schema definitions.

The transform logs the following events, with events marked with * causing the pipeline to fail:

- **SRIDAlreadyPresent**: Informational log entry indicating that the user defined an SRID while the input geometry already contains one.
- **IneligibleForPC***: A geometry that is incompatible with point coordinate representation (e.g., non-Point geometries or 3-/4-dimensional points) was processed while point coordinates were selected as the output format.
- **FailedToConvert***: A general error occurred during conversion due to faulty or invalid input data
- **UnknownInputField***: The user specified a non-existent field as input.
- **SRIDIncorrectlyFormatted***: The WKT input contains an SRID declaration, but it is incorrectly formatted.
- **GeometryIncorrectlyFormated***: This error occurs when the JTS readers throw a `ParseException`, indicating that the input geometry is incorrectly formatted.
- **UnableToFindInput**: This event occurs when Hop is unable to retrieve the field metadata from the preceding transform.

Figure 21 illustrates a sample use case for the Geometry Fields Converter. In this scenario, the WKB from the field 'geometry_wkb' is converted into WKT, while adding the SRID 2056 to all geometries. As the output field is left empty, this transform's output will contain an additional field, named 'geometry_wkt', containing the WKT representation of the geometries with the SRID 2056 prepended to all entries.

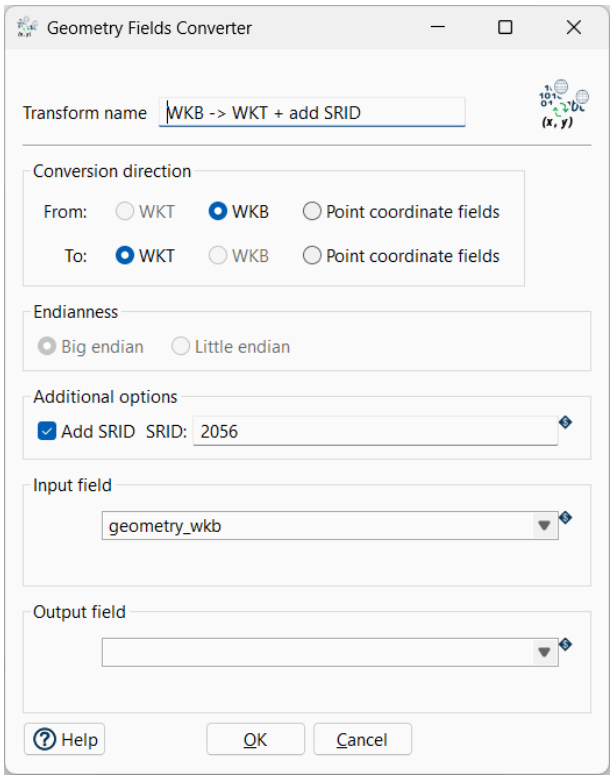

Figure 21: Example of a possible dialog window in practice.

For point coordinates, two fields are required as either output or input. Figure 22 shows an example of using point coordinates as input. The fields 'lon' and 'lat' will be treated as the x- and y-coordinates, respectively, of the resulting point geometry.

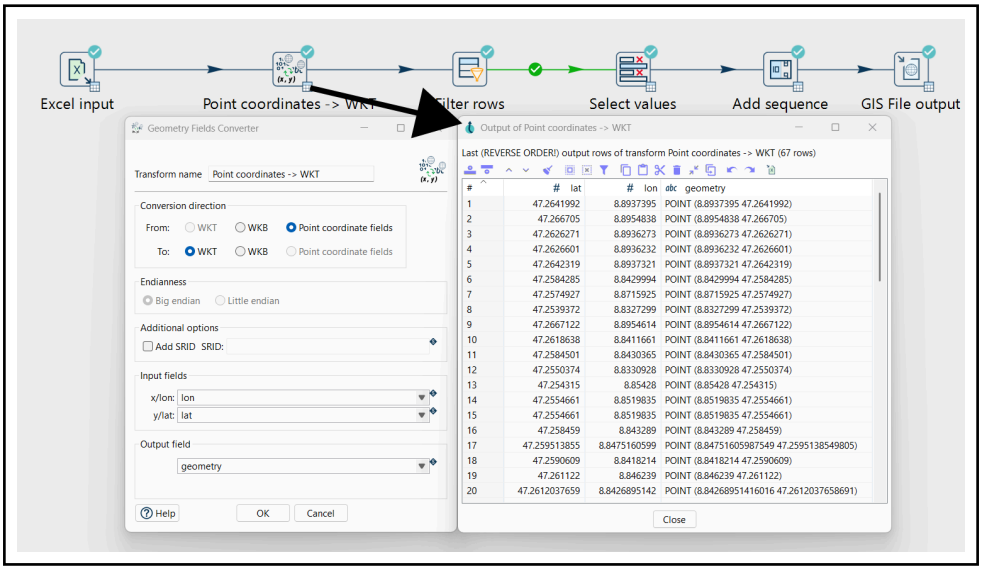

Figure 22: Example of point coordinates as input with resulting geometry in output.

Within the sample pipeline, an Excel file containing the coordinates of points of interest distributed across two columns (“lat” and “lon”) is read into the pipeline. Afterwards, the Geometry Fields Converter transform converts the point coordinate fields into WKT format, as described earlier.

The GIS plugins transforms are rather particular about their input; therefore, some input sanitisation is required. The GIS file output transform is unable to handle empty (`<null>`) entries, so these must be filtered out beforehand using a Filter rows transform. Additionally, the GIS plugins expect geometries to be stored as a field of type “Geometry”. To achieve this, as discussed in the Implementation Subchapter 2.4.3, a field type cast must be performed.

As we want to export our data as a GeoPackage, a primary key is required for each entry. This is accomplished by adding a sequential identifier to every row with the add sequence transform. Finally we can export our data as a GeoPackage, effectively having turned an Excel file into a GIS file.

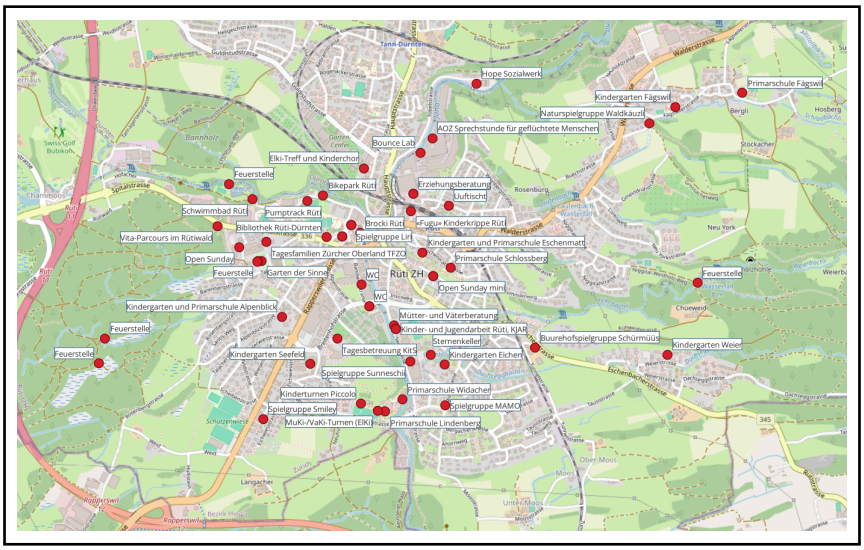

Figure 23: Resulting GeoPackage, viewed in QGIS.

2.4.2.1 Installation

To adhere to the non-functional requirement of installability (see Chapter 2.1.4.1), installation is made to be as trivial as possible and follows the same procedure as the OGR Vector Import/Export plugin (see Chapter 2.3.2.2). The plugin is provided as a ZIP file containing the transform’s JAR file along with the required JTS dependency, downloadable on the project’s releases page. After extracting the archive, the user simply copies the folder and its contents into the `hop/plugins/transforms/` directory.

A guide for manual installation is included in the project’s README.

2.4.2.2 Configuration

The transform offers the user to configure the conversion with the following options:

Conversion direction:

‘From’ defines the format of the input field(s). ‘To’ defines the desired output format.

Endianness:

In case 'WKB' is selected as the output format, the user can choose the byte order of the output binary. 'Big endian' is the default. In WKB, the first byte indicates the endianness of the following bytes. 00 for big endian, 01 for little endian.

Additional options:

Currently, only the option to prepend an SRID is available. This option is only enabled if the output format is either WKT or WKB. If the input already contains an SRID, it will be overwritten, and the user will be notified via a log entry in the Logging tab.

Input field(s): Defines the field(s) to be converted. If point coordinate fields are selected as input format, a second field containing the y-coordinate must be specified. The dropdown menu allows the user to select fields from the previous transform.

Output field(s):

The user may define the field in which the converted value will be stored. If the input format is point coordinate fields, a second output field may be specified. The fields may also be left empty, in which case new fields will be created according to the output format:
- WKT: 'geometry_wkt'
- WKB: 'geometry_wkb'
- Point coordinate fields: 'longitude', 'latitude'

The user may also define new fields by choosing unique names. A dropdown menu is available showing all fields from the previous transform. Selecting an existing field as output will overwrite the field and change the type.

2.4.3 Implementation

The Geometry class from the JTS library (*JTS Topology Suite*, 2016) is the main cornerstone of the transform. Every input is parsed into a Geometry object before converting it into the output format. This way, input validation is mostly already done by the WKB/WKT readers or for point coordinates fields, the GeometryFactory.

A limitation of the current implementation is, that it has no access to the Geometry ValueMeta by the GIS plugins. While the conversion from Geometry fieldtype to String fieldtype is straightforward, the transforms from the GIS plugins expect a Geometry ValueMeta as input and don't allow any fields of different types. This can be circumvented by changing the field type via the Select values transform (see Figure 24) and letting Hop handle the conversion internally. The reverse direction does not pose a problem: casting from *Geometry* to *String* is performed implicitly by Hop and works without issues.

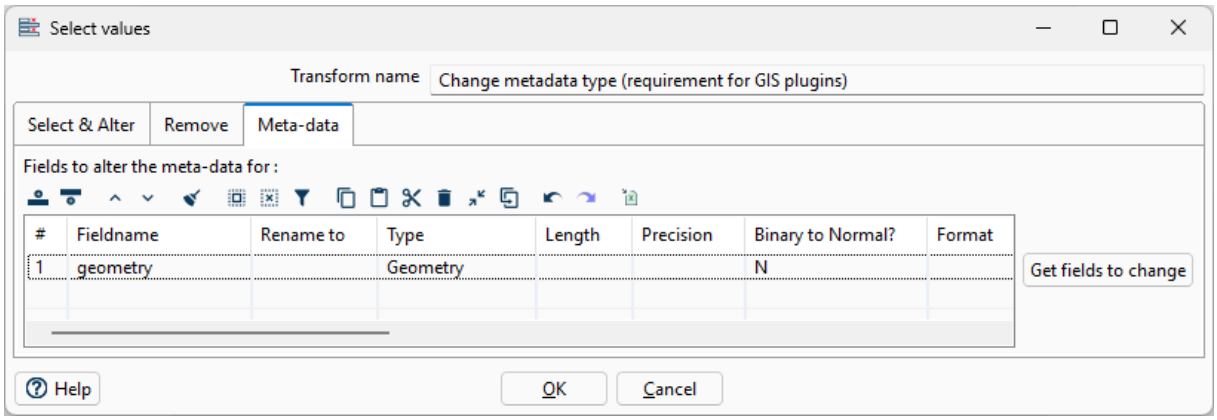

Figure 24: Select values transform with meta-data change for compatibility with the GIS plugins by Atol CD.

An option to save WKT as a *Geometry* field was planned but, due to the amount of time left in the project, not implemented.

Another noteworthy implementation detail is the change to using enums. Before the support for point coordinates was added, the conversion direction was represented by a simple boolean flag. This was not scalable, as adding more formats would increase the number of buttons needed exponentially. By using enums for the input and output formats, adding more formats is as simple as adding another enum value and implementing the corresponding conversion logic.

The transform is designed to implicitly handle SRID information, if present in the input WKB or WKT. The SRID is preserved during the conversion process and included in the output WKB or WKT. As SRID support for point coordinates fields would necessitate adding a third field, the team decided against implementing it.

2.5 Quality Measures

While non-functional requirements specify the desired quality attributes of the final product, it is equally important to define quality measures for the development process itself. This chapter outlines the guidelines, tools, and concepts used to maintain high quality throughout the project lifecycle. Ensuring an efficient and reliable development process is a key factor in delivering a high-quality product, making these measures crucial to the project's success.

2.5.1 Working Environment

Guidelines and definitions establish a structured framework for an efficient development process. The following definitions ensure a shared understanding of how tasks are to be carried out and when they meet the defined quality criteria.

2.5.1.1 Definition of Ready

To define when a user story is defined enough to be allowed to join the backlog, we have to establish a Definition of Ready. We settled on the well-established **INVEST** criteria:

- **I**ndependent: The story is independent of other user stories.
- **N**egotiable: The user story has been actively discussed among the team.
- **V**aluable: As this project is in the scope of a module, this is somewhat of a moot point. To accommodate this fact, we will instead count a story as valuable if it is demonstrable.
- **E**stimable: The effort of the story is estimable, so it can be planned in a sprint.
- **S**mall: (Ties in with E) The scope of a story should at max be 1 sprint iteration.
- **T**estable: The story has to be defined clearly enough as to make testing possible.

2.5.1.2 Definition Of Done

The Institute for Software has its own 'definition of done' for semester theses, which is applied here to determine when work items are deemed complete.

- Unit tests written and passed?
- Code review done?
- Documentation written or expanded?
- Source code committed to repository?
- Did the built version pass?
- Demo created and checked for product owner?
- OK received from product owner?

2.5.1.3 Documentation Guidelines

- Every team member is responsible for documenting their own work.
- Up to level three headers, a label is required. The following template should also be used:

```
=== Chapter Title <chapterTitle>

text
```

- References are to be used when referencing another chapter in the documentation: `@chapterLabel`.
- Keep the wording concise and grammatically as well as orthographically correct.
- Tables and figures/images all need a caption.
- Resources are to be saved in the `resources/` subfolder. Additional subfolders may be added inside.

- Cited sources must be credited via referencing their corresponding tag `@reference` in the bibliography. At a minimum, each entry requires a **title**, an **author** and/or **publisher** and a **publication date**. For online sources, it is also necessary to include the **access date** and the **URL**.
- AI tools may be used for conveniences like spellchecking or formatting.
- For the sake of consistency, only DeepL will be used for translations.
- The basic pattern provided by the template is to be applied. New files may be created if such a need arises.

2.5.1.4 Coding Guidelines

Since the OST does not provide any coding guidelines, we have to define them ourselves. We align ourselves with the Apache Hop code formatting guidelines, which falls in line with the Google Java Style Guide (*Google Java Style Guide*, 2025), as well as the provided books for the SEP2 lecture.

These guidelines are mainly enforced by the tools defined in the IDE chapter 2.5.2.1.

2.5.1.5 Time Tracking Guidelines

For a reliable overview of our time budget, it is imperative that the following guidelines are closely followed:

- Each work item must be linked to an epic, enabling monitoring of the time spent per epic.
- Logged time is rounded to the nearest 15 minutes for consistency.
- Time is tracked on the story level and not the subtask level.
- Work is logged as soon as possible.
- Task status is to be kept up to date.

2.5.1.6 Version Control Guidelines

All these rules apply to any kind of version control, be it documentation or code.

- Use feature branches following the scope of the given task.
- Feature branches at completion (see Definition of Done 2.5.1.2) will be merged into the main branch after a successful merge request.
- Basic version control rules apply as defined during the SEP1 module (keeping commits short, frequently fetch and push, etc.).
- Follow the process for merge requests.
- Each commit should be small enough to be described by one simple sentence.

2.5.2 Tools

Tools are essential in providing quality by aiding consistency and reliability throughout development. The following subchapter outlines the tools employed for development throughout the project.

2.5.2.1 IDE

For Java development, the team uses IntelliJ IDEA Ultimate. It supports industry-standard coding guidelines, as well as quick and easy building. Compared to Eclipse and Visual Studio Code, the team is most experienced with IntelliJ for Java development, which made it the natural choice for this project. To ensure consistency in the code for both team members, the default coding style settings of IntelliJ are used.

To enforce the Coding Guidelines (as defined in chapter 2.5.1.4), the google-java-format (*Google-Java-Format*, n.d.) formatter, a plugin for IntelliJ developed by Google, is used alongside IntelliJ's integrated linter combined with the Spotless plugin for Maven (*Spotless Maven Plugin*, 2025). The developer can check the source code for formatting issues and apply the fix with the command `mvn spotless:apply`.

2.5.2.2 CI/CD

Both plugins use GitLab CI/CD pipelines to perform automated tasks, including building the project, running tests, and checking code formatting. This provides developers with continuous quality assurance for every push and helps ensure the software remains intact during merge requests.

Both plugins share the same pipeline structure; however, the OGR Vector Import/Export plugin includes duplicate jobs for both action and transform. This isolates potential issues to one of the two components, making debugging easier.

The pipeline consists of the following stages:

- **build**
  - **check_formatting**: Runs `mvn spotless:check` to verify that the code adheres to the defined coding style.
  - **build**: Runs `mvn clean package` to build the project and save the resulting JAR files as artefacts.
- **test**
  - **unit_tests**: Runs the JUnit test suite using `mvn test` to verify that the business logic works as intended. (Only in the Geometry Fields Converter, see Chapter 2.5.3.1)
  - **integration_test**: Executes a basic predefined Hop pipeline to verify that the plugin integrates correctly with Hop and performs its intended tasks.
  - **speed_test**: Runs a basic predefined Hop pipeline to measure the speed of the action/transform (see Chapter 2.1.4.2) and saves the results as a JSON artefact.
  - **evaluate_performance**: Uses a simple bash script to process the JSON artefact from speed_test and evaluates whether performance meets the defined non-functional requirements, classifying it into the respective target categories.
  - **code_quality**: Runs the code quality template job (*Code Quality*, 2025) for a basic code review.
  - **semgrep-sast**: Runs the static application security testing (SAST) template job (*Static Application Security Testing (SAST)*, 2025) to perform a security scan on the repository.
- **release**
  - **prepare_release**: Triggered only when a new tag is pushed. Prepares a ZIP assembly containing the plugin's JAR file, dependencies (if needed), and a version XML file for easy drag-and-drop installation into Hop.

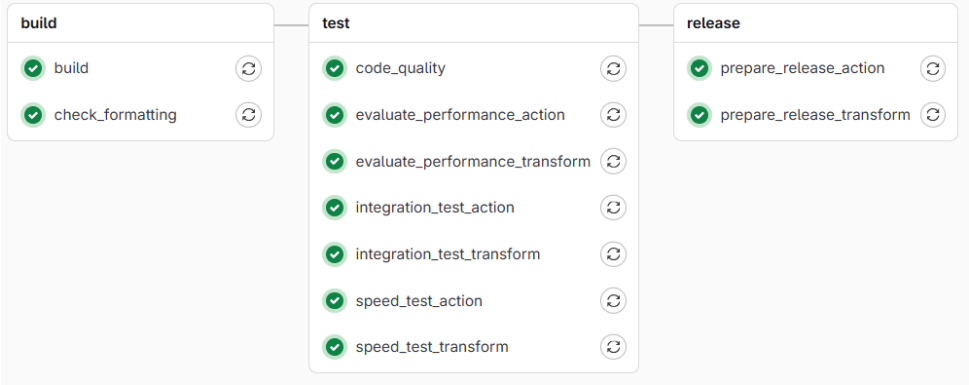

Figure 25: GitLab CI/CD pipeline triggered after a new tag is pushed to the OGR Vector Import/Export plugin.

Most jobs run in a Maven Docker container, with a few exceptions:

- Jobs executing Hop pipelines use the official Hop Docker container (*Docker Container*, 2025), which has Hop pre-installed. The exception is the OGR Vector Import/Export plugin, which requires GDAL to be pre-installed. In this case, manually installing Hop is faster, so a small Ubuntu GDAL by OSGeo Docker container (*Gdal Ubuntu-Small-Latest*, 2025) is used.
- Since Hop is written in Java 11, the check_formatting job (Spotless plugin) specifically uses an OpenJDK 11 Maven container.

2.5.2.3 AI Tools

The project follows the guideline “Umgang mit KI bei schriftlichen Arbeiten” (transl.: Dealing with AI in Written Work) issued by the School of Computer Science at OST, which permits and encourages the use of AI tools, provided that data protection requirements are met and that academic integrity, correctness, and proper referencing are ensured by the students.

AI tools used in this project include ChatGPT, DeepL and Grammarly. A complete list of aids is available in the appendix.

2.5.3 Test Concept

Testing is a fundamental method for ensuring software quality. Although successfully passing all tests does not guarantee defect-free software, it establishes a baseline demonstrating that the product satisfies stakeholder expectations. Increasing test coverage and test depth generally improves the likelihood that these expectations are met.

The following subchapter outlines all testing methods used for this project.

2.5.3.1 Unit Tests

To ensure the business logic functions correctly, a JUnit test suite is executed automatically with every push to the repository or manually via the `mvn test` command.

Only the Geometry Fields Converter plugin includes unit tests, as the OGR Vector Import/Export plugin primarily serves as a wrapper for the ogr2ogr command-line tool, which would require pure mock testing and provide limited value.

The Hop required classes (`Data`, `Meta` and `Dialog`) are not directly tested, even in official plugins, beyond mock testing or serialisation checks, or because they are user interface classes. Our group therefore decided that testing these classes is out of scope for this project.

2.5.3.2 Integration Tests

Integration tests verify the plugin's interaction with Hop. After each push, a predefined Hop pipeline/workflow is executed in the CI/CD pipeline using the Hop Docker container (*Docker Container*, 2025):

```
/opt/hop/hop-run.sh -j integration_test -r local -f /files/pipeline.hpl.
```

These tests ensure compatibility with Hop and other transforms/actions; any output is not validated.

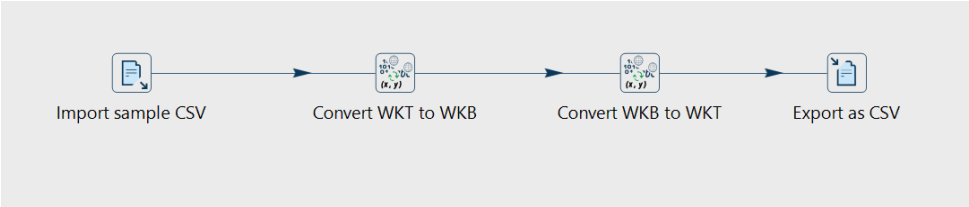

Figure 26: Integration test pipeline for the Geometry Fields Converter transform.

2.5.3.3 Acceptance Tests

During the transition phase, each functional and non-functional requirement is manually validated by the team to verify that they have been fully met. A requirement can either pass (✓) or fail (✗). The results can be reviewed in the Test Protocol of Chapter 2.5.3.4.

2.5.3.4 Test Protocol

ID	Description	Status
AHP-34	As a user, I want to convert the representation of geometry (WKT, WKB or point coordinates) of my GIS files interchangeably inside a Hop pipeline, so I can use them further in my pipeline.	✓
AHP-36	As a user, I want to convert my OGR format files into different OGR format files, so I can use them further in my pipeline/workflow.	✓
AHP-60	As a user, I want to keep the SRID of my geometries after conversions and have the option to add one.	✓
AHP-61	As a user, I want to define the byte order of my WKB-formatted geometry.	✓
AHP-64	As a user, I want to be able to add additional options to my ogr2ogr command.	✓
Total		5/5

Table 19: Test protocol for functional requirements.

ID	Description	Status
NFR #1	Installability	✓
NFR #2a	Scalability (Geometry Fields Converter)	Outstanding ✓
NFR #2b	Scalability (OGR Transform)	Minimal ✓
NFR #2c	Scalability (OGR Action)	Minimal ✓
NFR #3	Adaptability	✓
NFR #4	Open Source	✓
NFR #5	Interoperability	✓
NFR #6	Operability/Self-Descriptiveness	Outstanding ✓
Total		7/7

Table 20: Test protocol for non-functional requirements.

3 Project Documentation

The goal of this part is to provide an insight into the organisational matters of the project, such as planning and risk management.

3.1 Project Plan

During the inception phase of the project, an initial project plan was established. This plan serves as a baseline and is actively updated as needed.

3.1.1 Resources

The semester thesis guidelines dictate a total workload of at least 240 hours per student over the span of 14 weeks. It is the team's responsibility to distribute the workload evenly across the semester as best as possible. At the team's disposal lies an advisor with the objective to guide the team in the correct direction and act as the client stakeholder.

There are no financial means available; however, they are also not needed as the project requires no licences or access tokens.
Additionally, some members of the Institute for Software are available to support the development, enabling an exchange of expertise and pipelines. The institute will assume responsibility for maintaining the project after the completion of the semester thesis.

3.1.2 Processes

The project has been designed and planned according to a light application of Scrum+, which combines Scrum with the Rational Unified Process (RUP). Since the team consists of only two members, not all Scrum roles could be occupied. The Scrum ceremonies were instead scrapped and implemented more loosely to allow flexibility and allow closer work between the two members.

The 14-week span of the project is split into seven sprints, each lasting two weeks. Before each sprint, a set of tasks is defined, with the goal in mind to amount to a total of ~17 hours of work per member, to distribute work across the semester evenly.

3.1.3 Team Roles

Due to the size of our group, an extensive distribution of roles is not necessary; instead both members work on everything and are also responsible for everything. In line with agile principles, work items are assigned to members according to time budget and preference instead of skill. Where skill becomes a limiting factor, the other member is expected to help out.
Minor responsibilities such as keeping deadlines in mind, software architecture or DevOps will, however, be distributed.

Role	Assigned to	Description
Project Manager	Simon Weibel	Responsible for keeping deadlines in mind and submitting documents.
Lead Architect	Tobias Keel	Has the final say in architectural decisions.
Lead Developer	Tobias Keel	Has the final say in development-related decisions.
DevOps	Simon Weibel	Main responsibility is the CI/CD pipeline.

Table 21: Team Roles.

3.1.4 Meetings

Throughout the project, weekly meetings were held with the advisor, during which the team reported on progress since the previous meeting, discussed encountered challenges, and outlined plans for the upcoming period.

Meeting	CW / SW	Date	Time	Duration
Kickoff Meeting	38 / 1	19.09.2025	08:30	1 hour 30 minutes
Weekly Meeting 1	39 / 2	26.09.2025	08:30	1 hour 15 minutes
Weekly Meeting 2	40 / 3	03.10.2025	08:00	1 hour 15 minutes
Weekly Meeting 3	42 / 5	17.10.2025	09:00	1 hour 15 minutes
Weekly Meeting 4	44 / 7	31.10.2025	09:00	1 hour
Weekly Meeting 5	45 / 8	07.11.2025	10:00	2 hours
Weekly Meeting 6	47 / 10	21.11.2025	10:00	1 hour 45 minutes
Weekly Meeting 7	48 / 11	27.11.2025	13:00	1 hour
Q&A with Apache Hop Co-Founder	49 / 12	05.12.2025	10:00	1 hour
Weekly Meeting 8	50 / 13	12.12.2025	10:00	1 hour

Table 22: Meetings.

3.1.4.1 Apache Hop User Meetup

On 18 December, it-novum hosted the fifth Apache Hop user meeting, providing German-speaking Hop users the opportunity to follow presentations by Apache Hop co-founder Bart Maertens, Petra Stutz from the University of Salzburg, and our advisor, Prof. Stefan Keller. The accompanying chat enabled direct communication within the community and facilitated the exchange of questions and insights. This setting offered the team an excellent opportunity to clarify remaining Hop-related questions and engage with the broader user community.

3.1.4.2 Q&A with Apache Hop Co-Founder

In semester week 12, the advisor offered the team the opportunity to demonstrate the two plugins to Bart Maertens, co-founder of Apache Hop, in a meeting. During this session, the team was able to receive a general impression of the plugins and address specific Hop-related questions, such as extraction of performance metrics or the correct usage of the `Data` class.

3.1.5 Phases, Sprints, and Milestones

Scrum+ structures projects into phases and sprints, providing a broad guideline while maintaining the focus on the agile iterative process.

The following chapter explains the reasons behind dividing the project into **four phases** and **seven sprints** (or iterations), each lasting **two weeks**.

The subsequent graph serves as a visualisation of the timeline of the project, illustrating in which semester weeks (SW) each sprint (S*X*) occurs and how it aligns in its respective project phase.

SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8	SW9	SW10	SW11	SW12	SW13	SW14
15.9. - 21.9.	22.9. - 28.9.	29.9. - 5.10.	6.10. - 12.10.	13.10. - 19.10.	20.10. - 26.10.	27.10. - 2.11.	3.11. - 9.11.	10.11. - 16.11.	17.11. - 23.11.	24.11. - 30.11.	1.12. - 7.12.	8.12. - 14.12.	15.12. - 19.12.
S1 (15.9. - 28.9.)		S2 (29.9. - 12.10.)		S3 (13.10. - 26.10.)		S4 (27.10. - 9.11.)		S5 (10.11. - 23.11.)		S6 (24.11. - 7.12.)		S7 (8.12. - 19.12.)	
												Abstract	Submission
Inception		Elaboration				Construction						Transition	

Figure 27: Timeline of the project.

3.1.5.1 Phases

In the Rational Unified Process, a project is split into four phases to indicate its current state. These phases impose no strict constraints on when specific tasks must be performed, allowing for a smoother transition between them.
We decided to keep the distribution fairly standard, with the inception and transition phases each taking up one sprint, the construction phase taking up nearly half of the sprints with three, and the elaboration phase the remaining two sprints.

Inception	In the inception phase the team focuses on working out the task, creates the initial project plan and performs basic administrative tasks.
Elaboration	In the elaboration phase requirements are defined, prototypes are created
Construction	The construction phase focuses on the development of the software.
Transition	The transition phase marks the beginning of the end of development. Acceptance tests are performed, and the documentation gets finished.

Table 23: RUP phases.

3.1.5.2 Sprints

Sprints are a time-boxed window in which a set amount of work items is supposed to be finished, according to the Definition of Done Chapter 2.5.1.2.
As mentioned in Processes 3.1.2, the 14 weeks are split into seven sprints, each lasting two weeks. Since the project starts on Monday, 15 September, each sprint begins on Monday and ends on Sunday, with the sole exception of the last sprint, which ends on Friday, 19 December, at 17:00 due to the submission deadline.
The work items of a sprint are to be defined before the prior sprint ends. The sprints with their defined work items are viewable in the Jira Backlog.

3.1.5.3 Milestones

Milestones provide a clearly defined checklist to measure progress. We decided to orient ourselves according to the RUP phases, setting the end of a specific phase as the deadline for the respective milestone.

M1 - Inception
Date: 28.09.2025 (Sprint 1)
- Processes are defined
- Roles are distributed
- Long-term plan established

- Project workflow infrastructure is set up

M2 - Elaboration
Date: 26.10.2025 (Sprint 3)
- Coding infrastructure is set up
- Initial risks identified
- Functional requirements are defined
- Non-functional requirements are defined
- FRs and Non-FRs are planned appropriately
- Test concept established
- Initial architecture drafted
- A prototype is created

M3 - Geometry Fields Converter Plugin Complete
Date: 07.12.2025 (Sprint 6)
- All Geometry Fields Converter Plugin User Stories 2.1.3 are implemented

M4 - OGR Vector Import/Export Plugin Complete
Date: 07.12.2025 (Sprint 6)
- All OGR Vector Import/Export Plugin User Stories 2.1.3 are implemented

M5 - Construction
Date: 07.12.2025 (Sprint 6)
- Automated tests are set up
- Appropriate test coverage established
- Test concept finalised
- Architecture is adequate to software
- Milestones M3 and M4 passed
- All features are implemented

M6 - Transition
Date: 19.12.2025 (Sprint 7)
- Testing is finished
- Non-functional requirements are applied and tested
- Documentation is finished
- The abstract is submitted and accepted
- The tutorial is written

3.1.6 Schedule

The following graphic shows the rough planning of the project as epics. Each epic has a rough estimate of time.

		Inception	Elaboration		Construction			Transition
		Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	Sprint 7
	480h	SW 1-2	SW 3-4	SW 5-6	SW 7-8	SW 9-10	SW 11-12	SW 13-14
Project Management	140							
Project Planning	40							
Tooling / Infrastructure	40							
Testing	40							
Requirements Engineering	40							
Software Architecture	10							
WKB/WKT Converter Plugin	90							
ogr2ogr Plugin	80							
Total:	480							

Figure 28: Schedule.

3.1.6.1 Advisor Meetings

The advisor holds a dedicated meeting with the team each week or two to discuss what's been done, where the problems currently lie in and what's to be done until the next meeting. These meetings usually take place every Friday at 08:30 online or at the IFS office but can also be pre- or postponed.
These meetings are able to be skipped in weeks where there is not enough progress to show to justify a whole meeting.

3.1.6.2 Deadlines

For a project to succeed, deadlines must be met on time. For students, only two deadlines apply:

- Until **15 December**, enter the brochure abstract into the online tool and submit it to the advisor for review.
- Until **19 December 17:00**, submit the report to the advisor and upload all documents to the online tool.

3.1.7 Prototype

During the elaboration phase a prototype was developed as a proof of concept to address technical risks. Its purpose was to test the feasibility of the following points:

- Building a Plugin
- Adding a Plugin to Apache Hop
- Understanding the basic workings of Plugins

For this, the GIS Plugins by Atol CD (Atol CD, 2025), as well as the pre-installed Apache Hop plugins, were an important inspiration to get an initial grasp about the subject.

3.1.8 Planning Tools

The following tools supported the administration of the project, from documentation to issue tracking.

3.1.8.1 Time/Issue Tracking: Jira

Jira served as the central organisational platform for the project. All sprints, epics, user stories and tasks were defined and assigned, and the work time logged within Jira.

3.1.8.2 Repository: GitLab OST

The documentation's repository is hosted on the GitLab instance provided by the OST. It is part of the *Apache Hop-Plugins SA* group, which also hosts the two code repositories.
It contains an automatic build that compiles the Typst source files into a PDF file and is run after each push to the repository.

3.1.8.3 Microsoft Teams

Microsoft Teams served as the main communication channel for both the team and the advisor. It allowed project members to communicate quickly and efficiently, allowed the advisor to provide the team with immediate feedback on new findings, granted the ability to schedule and record meetings and helped keep the advisor up-to-date about progress and any issues encountered.

3.2 Risk Management

	Negligible	Minor	Moderate	Significant	Severe
Very Unlikely			R5		
Unlikely			R4	R3,R6	
Possible		R2		R1	
Likely			↑R5		↖R6
Very Likely					

Table 24: Initial Risk Matrix.

Number	R1
Name	Integration Issues
Risk	Different dependencies might be incompatible with each other or with the Apache Hop API.
Impact	Might need time-consuming workarounds or implementation of alternatives.
Mitigations	Build an early prototype that integrates all essential dependencies.

Table 25: Risk 1: Integration Issues.

Number	R2
Name	Testing Difficulty
Risk	Unit testing could turn out to be difficult because of Apache Hop framework dependencies.
Impact	Additional effort to set up (automated) testing.
Mitigations	-

Table 26: Risk 2: Testing Difficulty.

Number	R3
Name	Extended Illness
Risk	A team member might have an extended period of illness during the project and might not be able to contribute.
Impact	Not enough resources to complete all requirements.
Mitigations	Keeping documentation and Jira up to date so the other team member can seamlessly continue their work. Potentially reduce scope if necessary.

Table 27: Risk 3: Extended Illness.

Number	R4
Name	Poor Performance
Risk	Plugins might not be performant enough for large datasets.
Impact	Plugins might have limited usability for large use cases.
Mitigations	Integrate performance tests in the automated testing.

Table 28: Risk 4: Poor Performance.

Number	R5
Name	New Apache Hop Release
Risk	As version 2.16 of Apache Hop is set to be released before the end of the project, the team has to be prepared to make compatibility changes, as changes may break the plugins.
Impact	A significant amount of time may have to be allocated to fixing bugs in the plugins after release 2.16.
Mitigations	Keep up to date with changes for 2.16 by subscribing to Hop Commits and Developer Mailing Lists and add a time buffer to the project plan.

Table 29: Risk 5: New Apache Hop Release.

Number	R6
Name	Redundant Use Cases
Risk	As Apache Hop is a platform both team members do not have any experience with, there exists a possibility that features get discovered that may render existing use cases redundant.
Impact	The workload requirement of the semester thesis module might not be met.
Mitigations	Keep the project plan flexible to limit impact and define additional optional tasks as backup to mitigate.

Table 30: Risk 6: Redundant Use Cases.

3.2.1 Change Protocol

Risk	Date	Old Chance Old Impact	New Chance New Impact	Description
R6	07.11.2025	Likely Severe	Unlikely Significant	As during Sprint 4, the team once again came into a situation where one of the use cases stood at risk of turning redundant; the team had to react and mitigate and define a number of additional optional tasks in case another use case may render irrelevant at the discovery of a new feature.
R5	20.11.2025	Likely Moderate	Very Unlikely Moderate	Following the release of Apache Hop version 2.16 on 17 November and an early use case test session, no new defects were identified. Consequently, this risk was eliminated.

Table 31: Change protocol for the risk management matrix.

3.3 Problems

This chapter provides an in-depth description of the various problems encountered during the course of the project, as well as the corresponding solutions implemented to resolve them.

3.3.1 Discovery of Static Schema Definitions

During Sprint 3, the advisor informed the team about a Hop feature that could potentially render the CSV plugin use case redundant. Hop allows users to manually define static schema definitions as metadata for a project, which in turn enables the schema definition to be applied directly while reading the input (see Figure 29) for their location in Hop's GUI). If the input does not conform to the schema, `null` is insted returned, or a defined value in the case of `null`.

The advisor suggested that the team reconsider the CSV plugin, proposing a shift in focus: rather than solely validating CSV files, the plugin could allow users to import a schema into a static schema definition via a .csvs file. Instead of creating a transform solely for the CSV format, the plugin could validate a number of formats, such as JSON alongside CSV. The results should be documented in the form of an ADR (see Chapter 2.2.3).

Ultimately, as the team discovered the generic Data validator transform the static schema definitions lost relevance, and the concept of a dedicated CSV Validator transform was discarded (see Chapter 3.3.2). Although static schema definitions would indeed allow opening up the possibility to more formats, they do not allow the use of more elaborate features that the CSV Schema Language (*CSV Schema Language 1.2*, 2024) provides, such as regular expressions, and do not notify the user in case a file does not meet the schema's definitions.

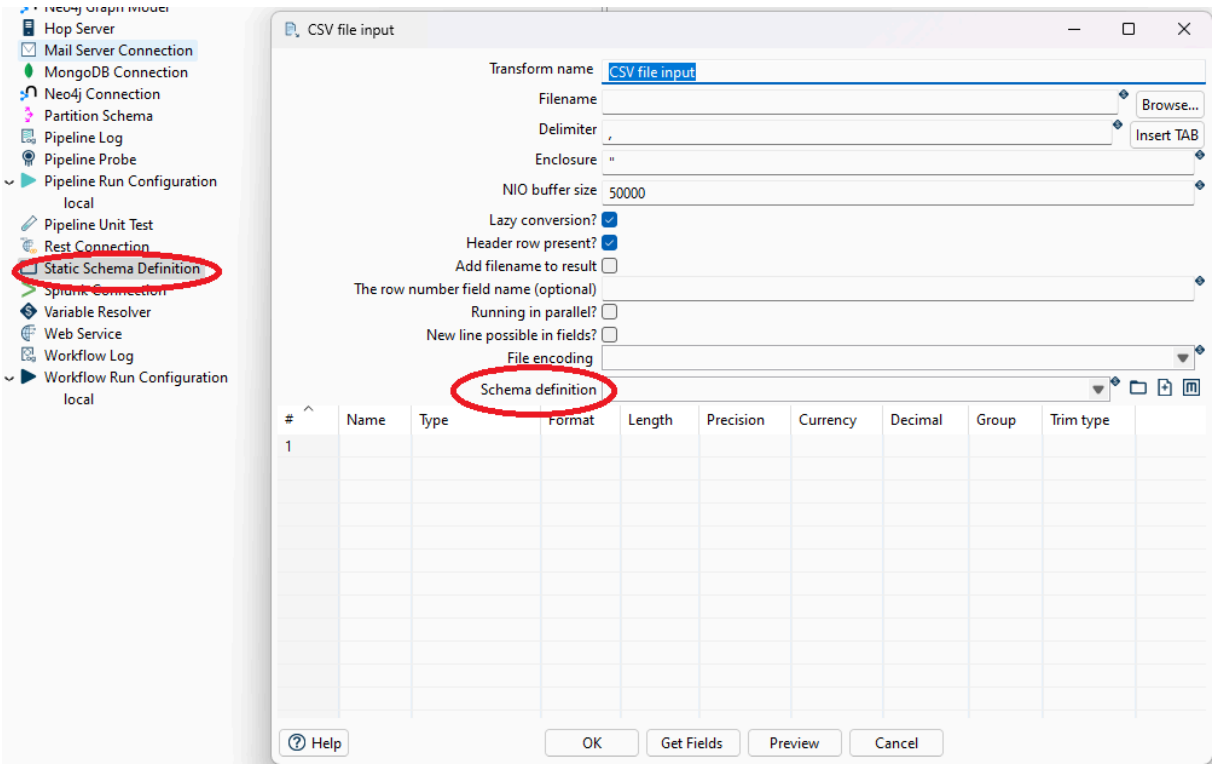

Figure 29: CSV file input dialog, showing the option to apply a schema definition to a file input. The project metadata panel on the left contains the static schema definitions.

3.3.2 Discovery of Generic Data Validator Transform

At the beginning of Sprint 4, the team discovered, through sheer coincidence, a generic Data validator that nearly mirrors all intended features of the CSV Validator. Although each field must be configured manually, barring metadata or semantic annotations such as foreign keys, the Data validator already implements all features defined by the CSV Schema Language (*CSV*

Schema Language 1.2, 2024) and even extends beyond them, being independent of file format, as it operates on a per-row basis (see Figure 30).

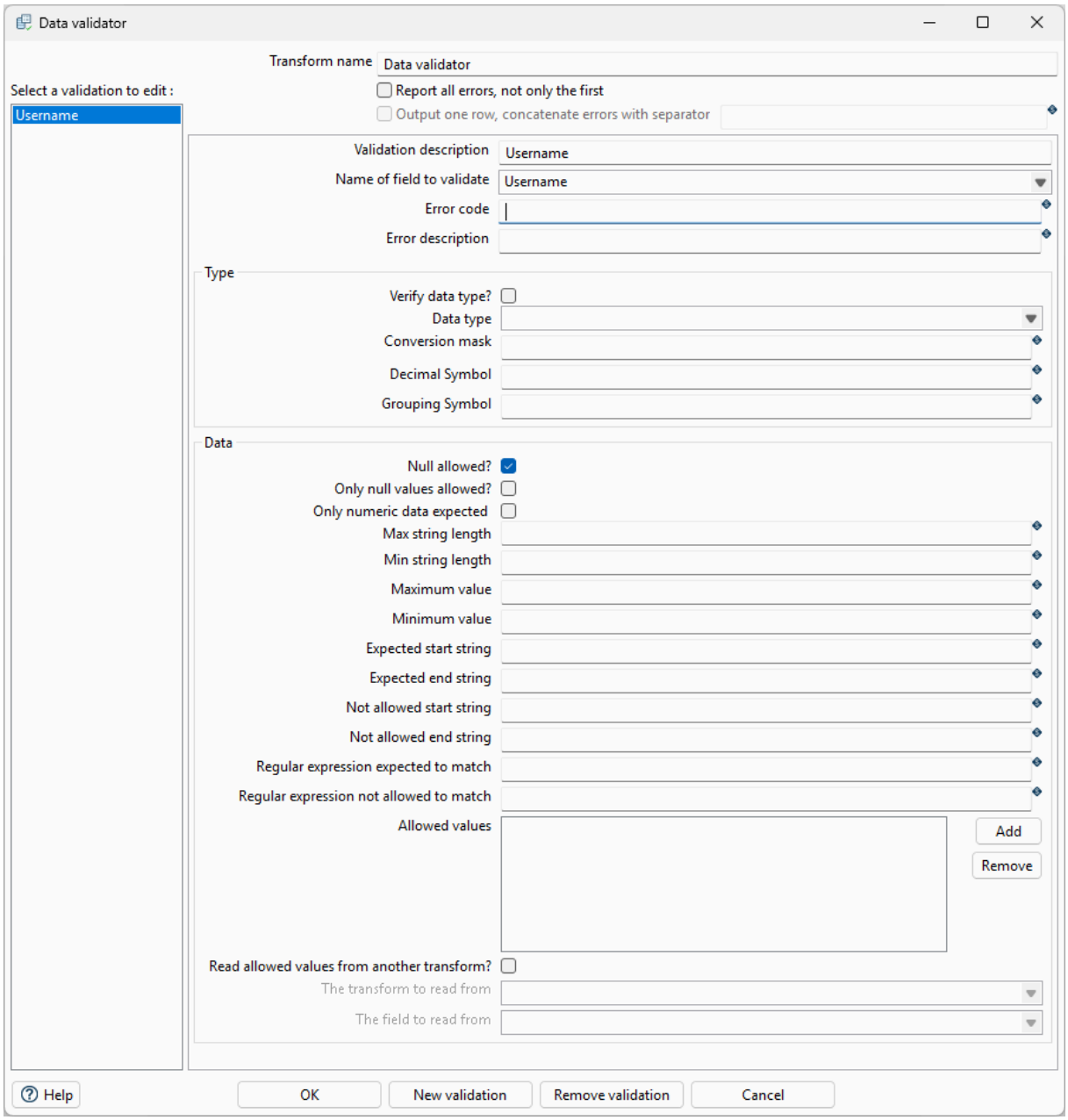

Figure 30: Data validator dialog form.

This posed a huge problem to the project, as half the work already done had been rendered obsolete and severe considerations had to be made. To accommodate for this situation, the advisor proposed another plugin: a transform to cast between the geometry representation forms of well-known text (WKT), well-known binary (WKB) and point coordinates, as such a function is currently missing in the Atol CD GIS plugins (Prof. Stefan F. Keller, 2025b).

The team agreed to this proposition at the next meeting and consequently set up plans to accommodate, as all the work done on the CSV Validator is not able to be carried over, meaning functional and non-functional requirements have to be redefined and the project plan has to be fully re-evaluated.

3.4 Conclusion

This semester thesis set out to extend Apache Hop with missing geospatial capabilities by developing two independent but complementary plugins: the ‘Geometry Fields Converter’ and ‘OGR Vector Import/Export’. Both plugins address concrete limitations of Apache Hop and the existing GIS plugins by Atol CD, particularly with regard to geometry representation handling and the limited number of supported GIS file formats.

The Geometry Fields Converter enables the conversion between common geometry representations such as Well-Known Text (WKT), Well-Known Binary (WKB), and point coordinate fields. By internally converting all inputs into JTS geometry objects, the transform ensures standard-compliant validation and preserves spatial metadata such as the SRID if applicable. An architectural decision was made to decouple the plugin from the Atol CD GIS plugins. Instead of relying on their custom geometry field type, compatibility is achieved through Hop’s existing metadata manipulation mechanisms like the ‘Select Values’ transform. This approach increases reusability, reduces coupling, and allows the transform to be used independently of any GIS-specific plugin ecosystem.

The OGR Vector Import/Export plugin integrates the powerful `ogr2ogr` utility into Apache Hop Pipelines and Workflows. By delegating format support to the locally installed GDAL instance, the plugin remains lightweight while supporting a wide range of vector formats far beyond Hop’s native capabilities. The implementation as both a transform and an Action allows flexible usage scenarios, from row-based batch conversions to workflow-controlled preprocessing steps. The structured handling of command-line options and robust error propagation ensure that advanced users retain full control over the conversion process without obscuring errors or introducing artificial limitations.

From a non-functional perspective, the plugins meet the defined requirements regarding scalability, interoperability, and platform independence. Performance measurements confirm that the Geometry Fields Converter scales efficiently with increasing data size, benefiting from Apache Hop’s inherent parallel execution model. While the OGR-based conversions are bound by external process execution and I/O constraints, the achieved performance still meets the defined minimal scalability goals. Both plugins integrate seamlessly into Apache Hop’s user interface and follow established design patterns, ensuring a familiar user experience.

Overall, this project demonstrates that Apache Hop can be effectively extended to support advanced geospatial workflows without compromising its architectural principles. The developed plugins close gaps in Hop’s GIS capabilities and provide a solid foundation for further geospatial integration within data orchestration pipelines.

3.4.1 Outlook

The following extensions and improvements could be considered for future development by the new maintainers at the Institute for Software.

3.4.1.1 OGR Vector Import/Export - Configurable error handling

Introducing configurable error handling for the **OGR Vector Import/Export** plugins, particularly for the transform, would significantly increase flexibility. For example, an optional ‘fail silently’ mode could be provided, allowing the Pipeline to continue execution even if individual OGR conversions fail. In such a scenario, only successfully converted files would be forwarded to the next transform, enabling more robust processing of large file collections.

3.4.1.2 OGR Vector Import/Export - Support for multiple input files

Extending the input configuration to allow the selection of multiple files via a file dialog would improve usability and reduce the need for auxiliary transforms such as 'Data Grid' in certain workflows. Comparable functionality already exists in the 'Microsoft Excel Input' transform. This enhancement would also increase the practical usefulness of the 'Append' behaviour of `ogr2ogr`, which is currently selectable but primarily beneficial when processing multiple input files.

3.4.1.3 Geometry Fields Converter - Native geometry field output

It is technically feasible to add native support for the geometry field type used by the Atol CD GIS plugins. Such an extension would need to be implemented carefully to avoid introducing a hard dependency on those plugins. Preliminary research suggests that this may be achievable using Apache Hop's existing data type infrastructure, although it could require a deeper analysis or partial reverse engineering of transforms such as 'Select Values', which are capable of handling all installed data types dynamically.

3.4.1.4 Geometry Fields Converter - JSON format support

A fringe but possible scenario is that geometries may be stored inside JSON objects generated within a pipeline, rather than imported from external files. The current implementation of the Geometry Fields Converter cannot parse or compose them. Adding support for JSON objects would be straightforward; this feature was omitted due to prioritisation and time constraints.

3.4.1.5 Atol CD GIS File Input - Field-based file paths

The current limitation of the Atol CD GIS file input transform to static file paths significantly restricts its usability in dynamic Pipelines. A preliminary implementation of field-based file input was already explored by Peter Bühler but proved unreliable and is not publicly available. A robust and well-tested implementation, contributed via a pull request to the Atol CD open-source repository, would greatly enhance the usefulness of the 'OGR Vector Import/Export' transform and improve interoperability within Apache Hop Pipelines.

3.5 Time Tracking Report

During the course of the project, time tracking was performed using the Jira work log feature to ensure that both members met the 240-hour requirement and to allow continuous assessment of the project's current status. Each member logged the time spent on individual tasks in 15-minute increments directly in Jira.

3.5.1 Total Time Invested

Figure 31 presents an overview of the total time spent on the project, surpassing the required 480 hours.

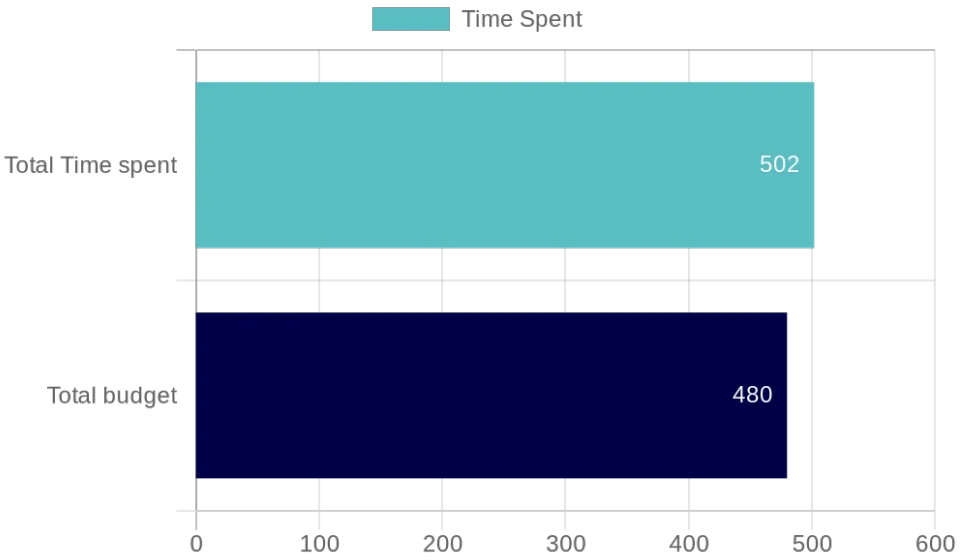

Figure 31: Total time spent over the course of the project.

3.5.2 Time Invested per Member

Figure 32 presents the time invested in the project by each team member.

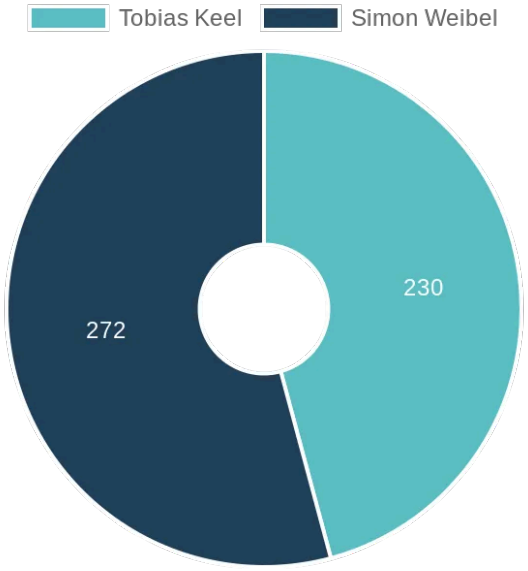

Figure 32: Total time spent per member.

A Appendix

The appendix contains all documents that are not essential for understanding the work but nevertheless offer added value and are sometimes mandatory for the semester thesis.

A.1 Module Description Semester Thesis Informatik

Excerpt from the module description at https://studien.ost.ch/ (*Modulbeschreibung - Studienarbeit Informatik*, 2023).

A.1.1 Key Points

Abbreviation	M_SAI21
Implementation period	HS/22-FS/26
ECTS	8
Learning objectives	The thesis should demonstrate the ability to solve problems using engineering methods. Accordingly, the thesis has a conceptual, theoretical and practical component.
Person responsible	Stocker Mirko
Location (offered)	Rapperswil-Jona
Prerequisite modules	SE Project (M_SEProj, FS/22-FS/25)
Additional prerequisite knowledge	none

Modulbeschreibung Semester Thesis Informatik.

A.1.2 Description

The Semester thesis is usually worked on in teams of two and supervised by a lecturer. The semester thesis lasts 14 weeks with a total workload of at least 240 hours per person. The topic of the thesis can be suggested by an external company, the lecturer or the students. The final assignment is set by the supervising lecturer. The thesis should demonstrate the ability to solve problems using engineering methods. Accordingly, the thesis should include a conceptual, theoretical and practical component. As a rule, a computer science project is worked on with the following subtasks:

- Familiarisation with a new task
- Planning of the project
- Analysis of requirements (including description of the environment and delimitation)
- Design and implementation of the solution (including evaluation of the state of the art)
- Testing of the solution
- Evaluation and outlook

B Glossary

Apache Hop	
Pipeline	A network of transforms, connected by Hops and together with workflows, are the main building blocks in Hop.
Transform	Modular components used to process and transform data, they are the building blocks that are used to create a pipeline.
Workflow	Orchestrate pipelines via a series of actions and/or pipelines for dataflow management.
Action	Discrete workflow components that perform sequential tasks, such as file operations, job control, or pipeline execution.
Hop	A directed connection between transforms or actions in a pipeline/ workflow, representing the flow of data.
Plugin	Modular components that extend Hop's core functionality without altering its source code. Their size is not fixed; a plugin may consist of a single transform or include multiple actions, transforms, and associated metadata.

Git	
Git	A distributed version control system for tracking changes in source code.
GitLab	A web-based DevOps platform providing version control, issue tracking, and CI/CD.
Commit	An object containing recent changes.
Branch	A graph of commits.

Pipeline	Automated process with a set of defined jobs. Usually performed every push to origin.
CI/CD	(Continuous Integration and Continuous Delivery) The practice of streamlining building, testing and deploying software quickly and regularly.

Project Management

Scrum	Agile, iterative project management framework mainly defined via roles, events and a time-boxed iteration cycle. Roles include the Product Owner, Scrum Master and Development Team, while key events are the Daily Scrum, Sprint Review and Sprint Retrospective.
Rational Unified Process (RUP)	An iterative software development process that splits a project into four life-cycle phases.
Sprint	A time-boxed development iteration, usually lasting 1-4 weeks, during which a planned set of tasks is implemented, tested, and reviewed, producing a shippable product increment. In this project, the term is used interchangeably with Iteration.

Software Development

Long-Term Support (LTS)	A software release that receives extended maintenance, including security updates, bug fixes, and critical patches, for an officially defined period, ensuring stability and reliability for production use.
Linter	Tool part of an integrated development environment that helps enforcing coding guidelines/rules either directly or by notifying the developer.
Factory	A design pattern used to create objects without specifying the exact class of object to create.

Geographic Information Systems

QGIS	Open-source geographic information system tool.

Geospatial Data Abstraction Library (GDAL)	Ppen-source library for reading, writing, and converting raster and vector geospatial data across multiple formats.
ogr2ogr	Part of GDAL. Command line utility used to convert from and to a wide range of different GIS file formats.
Well-Known Text (WKT)	A text-based format for representing geometric shapes in a human-readable form.
Well-Known Binary (WKB)	A binary format for representing geometric shapes in a compact manner.
Point Coordinate	A geometric representation of a point defined by an x and y value in a two-dimensional coordinate system
Spatial Reference Identifier (SRID)	A numeric identifier that uniquely identifies a spatial reference system.
European Petroleum Survey Group Geodesy (EPSG)	An international standard that defines and maintains codes for coordinate reference systems, datums, and map projections. e.g. EPSG:2056: CH1903+ / LV95, the standard Swiss national coordinate reference system.
Point of Interest (POI)	Represents a specific, identifiable location in the real world, often annotated with descriptive metadata, and is commonly modeled as a single point geometry in geospatial applications.

File Formats

Scalable Vector Graphics (SVG)	A vector image format for two-dimensional graphics with support for interactivity and animation.
Comma-Separated Values (CSV)	A simple file format used to store tabular data, such as a spreadsheet or database.
GeoPackage (GPKG)	An open, non-proprietary, platform-independent and standards-based data format for geographic information system implemented as a SQLite database container.

ESRI Shapefile (SHP)	A popular geospatial vector data format for geographic information system software.
JavaScript Object Notation (JSON)	A lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate.
GeoJSON	A format for encoding a variety of geographic data structures using JavaScript Object Notation (JSON).
File Geodatabase	a proprietary, file-based spatial database format developed by Esri for storing, managing, and analyzing geospatial data.

C List of Tables

D List of Figures

E List of Aids

Area of Responsibility	Tools
Collaboration and project management	• Microsoft Teams • Jira
Coding	• ChatGPT • IntelliJ IDEA Ultimate • Visual Studio Code • Git • GitLab • Maven • Java
Data analysis and visualisation	• QGIS • Microsoft Excel • Apache Hop • Notepad++ • Inkscape
Documentation	• Visual Studio Code • Typst • Git
Translation	• DeepL
Text creation, text optimisation, spelling and grammar checking	• ChatGPT • Grammarly • DeepL • QuillBot
Prototyping	• Microsoft Paint
DevOps	• GitLab • Docker • Maven

F Bibliography

Apache Hop. (2025, November 19). Apache Software Foundation. https://hop.apache.org/

Apache Hop Architecture. Apache Software Foundation. Retrieved December 16, 2025, from https://hop.apache.org/docs/architecture/

Apache Software Foundation. (2021, October 5). *Project Hop GitHub*. GitHub. https://github.com/project-hop

Apache Software Foundation. (2022, December 14). *Apache Hop plugins Samples*. GitHub. https://github.com/project-hop/hop-plugin-sample

ASF Source Header and Copyright Notice Policy. (2004). Apache Software Foundation. https://www.apache.org/legal/src-headers.html

Atol CD. (2025, June 4). *Apache Hop GIS Plugins*. GitHub. https://github.com/atolcd/hop-gis-plugins/tree/master

Code Quality. (2025, November 18). GitLab. https://docs.gitlab.com/ci/testing/code_quality/

CSV Schema Language 1.2. (2024, June 4). The National Archives (UK). https://digital-preservation.github.io/csv-schema/csv-schema-1.2.html

Docker container. (2025, October 1). Apache Software Foundation. https://hop.apache.org/tech-manual/latest/docker-container.html

Eclipse Foundation. (2025, October 28). *SWT: The Standard Widget Toolkit*. Eclipse Foundation. https://eclipse.dev/eclipse/swt/

gdal ubuntu-small-latest. (2025, December 14). Open Source Geospatial Foundation. https://github.com/osgeo/gdal/pkgs/container/gdal

Google Java Style Guide. (2025, October 1). Google. https://google.github.io/styleguide/javaguide.html

google-java-format. JetBrains Marketplace. https://plugins.jetbrains.com/plugin/8527-google-java-format

ISO/IEC 25010:2023. (2023). International Organization for Standardization. https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25010:ed-2:v1:en

JTS Topology Suite. (2016, November 4). Eclipse Foundation. https://github.com/locationtech/jts

Modulbeschreibung - Studienarbeit Informatik. (2023). OST – Ostschweizer Fachhochschule. https://studien.ost.ch/allModules/40906_M_SAI21.html

Olaf Zimmermann. (2020). *Y-Statements*. Medium. https://medium.com/olzzio/y-statements-10eb07b5a177

Prof. Stefan F. Keller. (2025a, October 23). *[Feature] Add "Wait for completion" option to "GIS file input" (and output) transform #30*. https://github.com/atolcd/hop-gis-plugins/issues/30

Prof. Stefan F. Keller. (2025b, November 3). *[Feature] Add a transform (e.g. “Select GIS Value”) for casting between GIS transforms geometry type and WKT/WKB/bytea #31*. https://github.com/atolcd/hop-gis-plugins/issues/31

Projects & Environments. (2025, April 3). Apache Software Foundation. https://hop.apache.org/manual/latest/projects/index.html

Spotless Maven Plugin. (2025, November 26). DiffPlug. https://github.com/diffplug/spotless/tree/main/plugin-maven

Static application security testing (SAST). (2025, December 12). GitLab. https://docs.gitlab.com/user/application_security/sast/

Vector drivers. (2025, August 31). Open Source Geospatial Foundation. https://gdal.org/en/stable/drivers/vector/index.html