

Jade Hochschule Oldenburg

Fachbereich: Bauwesen Geoinformation Gesundheitstechnologie

Studiengang: Geoinformationswissenschaften



Der deutsche Standard XPlanung in den Gemeinden und Städten.

Machbarkeitsstudie zur Umsetzung eines Exports nach XPlanGML mit QGIS.

The German standard XPlanung in municipalities and cities.

Feasibility study for the implementation of an export to XPlanGML with QGIS.

Masterarbeit zur Erlangung des akademischen Grades

„Master of Science“ (M. Sc.)

|                  |   |
|------------------|---|
| Eingereicht von: | Lukas Hermeling<br>Modersohn-Becker-Hof 8<br>49716 Meppen<br>E-Mail: lukas.hermeling@web.de<br>Matrikelnummer: 6016692<br>Fachsemester: 6 |
| Erstprüfer:      | Prof. Dr. Roland Pesch<br>Jade Hochschule Oldenburg<br>Ofener Str. 16/19<br>26121 Oldenburg   |
| Zweitprüfer:     | Prof. Dr.-Ing. Sebastian Hollermann<br>Jade Hochschule Oldenburg  |
| Eingereicht am:  | 30. September 2022  |

## Selbstständigkeitserklärung

Name: Hermeling

Vorname: Lukas

Matrikelnummer: 6016692

Erklärung gemäß § 18 (7) Allgemeiner Teil (Teil A) der Prüfungsordnung für die  
Master-Studiengänge (MPO) an der Jade Hochschule  
Wilhelmshaven/Oldenburg/Elsfleth  
in der Fassung der Bekanntmachung vom 08. August 2017 (VkBl. Nr. 90/2017)

Die Master-Arbeit ist

eine Einzelarbeit.

eine Gruppenarbeit zusammen mit der/dem Studierenden:

\_\_\_\_\_

Ich versichere hiermit, die Master-Arbeit

bei einer Gruppenarbeit den/die Teil(e)

\_\_\_\_\_

selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Zudem versichere ich, alle Stellen der Arbeit, die wortwörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht zu haben und die Arbeit - mit Ausnahme für einen Double oder Joint Degree - in gleicher oder ähnlicher Form noch keinem anderen Prüfungsverfahren im In- oder Ausland zugrunde gelegen hat bzw. als Studienabschlussarbeit an keiner anderen Hochschule eingereicht wurde.

\_\_\_\_\_

(Ort, Datum)

\_\_\_\_\_

(Unterschrift Studierende/r)

## Zusammenfassung

Der deutsche Standard XPlanung ist ein Planungsstandard, der von den deutschen Kommunen bis in das 2022 umgesetzt werden muss. Da sich die deutschen Gemeinden und Städte im stetigen Wandel befinden, soll durch einen Standard die Effizienz der Planungsprozesse gesteigert werden. Damit dieser Standard auch in Kommunen mit geringem Budget umgesetzt werden kann, müssen die Open Source Produkte für die einwandfreie Nutzung angemessen programmiert sein. Somit beschäftigt sich diese Arbeit mit der Weiterentwicklung des Open Source Produktes QGIS und dem hier enthaltenen Plugins XPlanung, da in dem bestehenden Plugin noch einige wichtige Funktionalitäten fehlen. In dieser Arbeit wird die Umsetzbarkeit der Entwicklung einer Exportschnittstelle für den objektorientierten Datenaustauschstandard XPlanGML untersucht, der nicht nur für den Austausch der Pläne, sondern auch für die Digitalisierung des Bauantrags relevant ist. Hierbei wird ein Überblick über den Standard XPlanung sowie die Umsetzung in der Open Source Software QGIS gegeben. Anhand des vorhandenen Plugins wird die Funktionalität der Exportschnittstelle für XPlanGML-Dateien in die vorhandene Struktur integriert. Bei der Entwicklung werden verschiedene Bibliotheken und Software für die Entwicklung des Exports analysiert. In Rahmen dieser Arbeit wird dann eine prototypische Exportschnittstelle für einige XPlanung-Objekte der Bebauungspläne umgesetzt. Aus dieser Erzeugung der XPlanGML-Daten wird ein Schema abgeleitet, an Hand dem die noch fehlenden XPlanung-Objekte für die Bebauungspläne und die weiteren Planwerke weiterentwickelt werden können. Abschließend wird das Ergebnis der XPlanGML-Datei mit dem XPlanValidator untersucht und auf gewisse Problematiken in der Entwicklung der Exportschnittstelle eingegangen.

# Inhaltsverzeichnis

|  |     |
|--|-----|
| Zusammenfassung.....                                 | II  |
| Inhaltsverzeichnis.....                              | III |
| Abbildungsverzeichnis.....                           | V   |
| Abkürzungsverzeichnis.....                           | VI  |
| 1 Einleitung .....                                   | 1   |
| 1.1 Ziel der Arbeit und Forschungsfrage .....        | 2   |
| 1.2 Struktur der Arbeit.....                         | 3   |
| 2 Fachliche Grundlagen .....                         | 5   |
| 2.1 Digitalisierung von Bebauungsplänen .....        | 5   |
| 2.1.1 Bebauungspläne .....                           | 5   |
| 2.1.2 XPlanung.....                                  | 6   |
| 2.2 Umsetzung von Bebauungsplänen mit GIS .....      | 12  |
| 2.2.1 Geoinformationssystem (GIS) .....              | 12  |
| 2.2.2 QGIS.....                                      | 13  |
| 2.2.3 Plugin XPlanung.....                           | 14  |
| 3 Material und Methodik.....                         | 16  |
| 3.1 Software und Bibliotheken .....                  | 16  |
| 3.1.1 Hale Studio .....                              | 16  |
| 3.1.2 GDAL/OGR-Bibliothek .....                      | 18  |
| 3.1.3 lxml-Bibliothek.....                           | 19  |
| 3.1.4 pygml-Bibliothek .....                         | 20  |
| 3.1.5 IO-Modul .....                                 | 21  |
| 3.2 Methodik und Durchführung.....                   | 22  |
| 3.2.1 Struktur des vorhandenen Plugins XPlanung..... | 22  |
| 3.2.2 Anforderungen an den XPlanGML-Export.....      | 25  |

|       |   |    |
|-------|---|----|
| 3.2.3 | Prototypische Entwicklung des Tools zum XPlanGML-Export ..... | 28 |
| 4     | Ergebnisse .....  | 45 |
| 5     | Diskussion .....  | 51 |
| 6     | Fazit und Ausblick .....                                      | 58 |
|       | Literaturverzeichnis.....                                     | 60 |
|       | Anhang.....   | 66 |

## Abbildungsverzeichnis

|   |    |
|---|----|
| Abbildung 1: Zusammenarbeit bei dem Planungsprozess ohne (links) und mit (rechts) dem XPlanung-Datenaustauschstandard. ....       | 8  |
| Abbildung 2: Vereinfachte Struktur des XPlanung-Datenmodell. ....   | 10 |
| Abbildung 3: Beispiel für eine Data-Driven Input Mask für das XPlanung-Plugin. ....   | 14 |
| Abbildung 4: Benutzeroberfläche von hale Studio mit dem Versuch des Exports von XPlanGML-Dateien. ....                            | 17 |
| Abbildung 5: Codebeispiel für die pygml-Bibliothek.....   | 21 |
| Abbildung 6: Beispiel für das Schreiben eine Textdatei mit den Funktionen von dem io-Modul. ....                                  | 22 |
| Abbildung 7: Übersicht der Python-Skripte mit den Klassen zu dem QGIS-Plugin XPlanung..   | 23 |
| Abbildung 8: MVC-Architektur mit den Abhängigkeiten. ....   | 24 |
| Abbildung 9: Auszug in Notepad++ aus der XPlanGML_BPlan.xsd für die Klasse BP_Bereich.  | 27 |
| Abbildung 10: UML-Aktivitätsdiagramm des Tools Exportieren in QGIS.....   | 29 |
| Abbildung 11: GUI für den Export von XPlanGML-Dateien aus QGIS. ....  | 31 |
| Abbildung 12: GUI für die Auswahl des Plangebiet für den Export. ....   | 32 |
| Abbildung 13: Ausschnitt aus QGIS mit dem erzeugten Bebauungsplan mit dem XPlanung-Plugin in Spiekeroog.....                      | 45 |
| Abbildung 14: Ausschnitt des XPlanGML-Ergebnisses für den Export des Bebauungsplans von Spiekeroog in Notepad++.....              | 46 |
| Abbildung 15: Ergebnis der Validierung für die XPlanGML-Datei des Bebauungsplans in Spiekeroog.....                               | 48 |
| Abbildung 16: Kartenvorschau in dem Validierungsbericht. ....   | 49 |
| Abbildung 17: Import der Ergebnis-XPlanGML in ArcGIS in das Tool GeoOffice XPlanung.....  | 50 |
| Abbildung 18: Exportschnittstelle für XPlanGML-Dateien in ArcGIS aus der Erweiterung GeoOffice xPlanung.....                      | 55 |
| Abbildung 19: Integration und Visualisierung der antragsrelevanten Daten für einen Bauantrag in der prototypischen Software. .... | 57 |

## Abkürzungsverzeichnis

|               |  |
|---------------|--|
| <b>API</b>    | Application Programming Interface (Programmierschnittstelle) |
| <b>CAD</b>    | Computer Aided Design  |
| <b>BauGB</b>  | Baugesetzbuch  |
| <b>BauNVO</b> | Baunutzungsverordnung  |
| <b>GDAL</b>   | Geospatial Data Abstraction Library                          |
| <b>GIS</b>    | Geoinformationssysteme                                       |
| <b>GML</b>    | Geography Markup Language                                    |
| <b>GMLAS</b>  | Geography Markup Language driven by application schemas      |
| <b>GPL</b>    | General Public License                                       |
| <b>GUI</b>    | Graphical User Interface                                     |
| <b>ISO</b>    | International Organization for Standardization               |
| <b>JSON</b>   | JavaScript Objekt Notation                                   |
| <b>KIT</b>    | Karlsruher Institut für Technologie                          |
| <b>MUR</b>    | Minimal umgebendes Rechteck                                  |
| <b>NAS</b>    | Normbasierte Austauschchnittstelle                           |
| <b>OGC</b>    | Open Geospatial Consortium                                   |
| <b>PlanZV</b> | Planzeichenverordnung  |
| <b>XML</b>    | Extensible Markup Language                                   |
| <b>XSD</b>    | XML Schema Definition  |
| <b>XÖV</b>    | XML der öffentlichen Verwaltung                              |

# 1 Einleitung

Da sich die deutschen Gemeinden und Städte im stetigen Wandel befinden, müssen sich die Behörden auf diese sich wandelnde Struktur einstellen. Die damit einhergehenden Veränderungen machen sich mit den entsprechenden Baumaßnahmen in den Kommunen bemerkbar. Um diese Veränderungen zu kontrollieren, müssen Pläne in den Behörden der Gemeinden und Städten rechtzeitig aufgestellt werden. In den Planungsprozessen in den deutschen Behörden wird immer mehr auf einen standardisierten Ablauf der Verfahren hingearbeitet. Dies soll zu Kostenreduzierungen, einen schnelleren Verfahrensablauf, die Verminderung von Datenverlusten im Verfahren und weiteren Vorteilen führen. Dafür wurde vom IT-PLANUNGSRAT (2017) der deutsche Standard XPlanung für alle Gemeinden verabschiedet. Dieser besagt, dass alle deutschen Kommunen den Standard XPlanung in den Bereichen der Bauleitplanung, der Landschaftsplanung, der Raumordnung und sonstige raumbezogene Planwerke bis 2022 umsetzen müssen. Dies ist gerade für kleinere Kommunen mit einem geringen Budget eine schwierige Angelegenheit. Da der Standard XPlanung in verschiedenen Softwareprodukten umgesetzt wurde, muss eines dieser Produkte in den Gemeinden angeschafft werden. Weil dies immer eine Frage des Budgets ist, kann hier nicht unbedingt eine neue kommerzielle Software angeschafft werden. Somit muss auch der Bereich der Open Source Produkte in Betracht gezogen werden. Da in den Planwerken mit verschiedenen raumbezogenen Informationen auf unterschiedlichen Ebenen gearbeitet wird, ist es sinnvoll diese Arbeiten mit einem Geoinformationssystem (GIS) durchzuführen (BILL 2010, S. 662 f).

Im Hinblick dieser Punkte stoßen die kleineren Kommunen mit einem geringeren Budget auf die Open Source GIS-Software QGIS. In dieser Software wurde der Planungsstandard XPlanung von STRÖBL (2021a) mit Hilfe von einem Plugin realisiert. Das Plugin enthält verschiedene Tools wie die Erzeugung von den Planwerken, den Import von XPlanGML-Dateien und weitere Funktionalitäten. Zudem gibt es eine Liste mit Tools, die noch nicht realisiert wurden. Darunter fällt zum Beispiel das Tool des Exports von XPlanGML-Dateien. Um für die Kommunen ein vollständiges Produkt für den Standard XPlanung in einer GIS-Schale zu schaffen, muss dieses Plugin um die fehlenden Tools erweitert werden. Bei einem unvollständigen Open Source Tool müssen die Gemeinden und Städte mit einem geringeren Budget zwischen einem GIS-



Spezialisten, der mit einem unvollständigen Produkt umgehen kann, oder einem vollständigen kommerziellen Produkt abwägen.

Aus diesem Grund beschäftigt sich diese Arbeit mit der Erweiterung des Open Source QGIS Plugins XPlanung. Dabei soll in dieser Arbeit mit einer Machbarkeitsanalyse untersucht werden, ob ein Export von XPlanGML-Dateien über Bibliotheken realisiert werden kann. Somit könnte das Plugin um ein sehr wichtiges Tool erweitert werden. Mit dieser Tool-Erweiterung könnte der Anreiz für Kommunen größer werden, das Open Source Produkt QGIS zu nutzen, um bei der Umsetzung im Planungsprozess mit dem deutschen Standard XPlanung Geld einzusparen.

## 1.1 Ziel der Arbeit und Forschungsfrage

Da die Entwicklung der Planungsprozesse immer weiter digitalisiert und neue Standards für die Durchführung aufgestellt werden, müssen auch die Softwareprodukte, in denen die Standards umgesetzt werden, mit einem entsprechenden Funktionsumfang ausgestattet sein. Dabei werden oft verschiedene kommerzielle Produkte angeboten. Aber gerade bei den Open Source Produkten in diesen Bereich muss die Weiterentwicklung vorangetrieben werden. Es gibt in vielen deutschen Gemeinden und Städten das Problem, dass diese über ein geringes Budget verfügen und somit genau schauen müssen wofür dieses eingesetzt wird. Dabei ist ein wichtiger Punkt, dass bei Softwareprodukten auf Open Source Lösungen gesetzt werden kann. Hierbei muss beachtet werden, dass viele dieser Lösungen meist nicht vollständig ausgereift sind, da sie von Organisation und Entwicklern nicht hauptberuflich entwickelt werden.

Somit soll mit dieser Arbeit die Erweiterung der Open Source Software QGIS mit dem Plugin XPlanung vorangetrieben werden. Dabei soll mittels einer Machbarkeitsanalyse untersucht werden, ob eine Exportschnittstelle für XPlanGML-Dateien entwickelt werden kann. Auf der Plattform GitHub auf dem das Tool von STRÖBL (2021a) veröffentlicht wurde, sind Anfragen von Nutzern bezüglich des benötigten Exports zu sehen. Somit ist ersichtlich, dass ein wichtiges Tool in dem Plugin noch fehlt.

Die folgenden Forschungsfragen sollen dabei in dieser Arbeit beantwortet werden:

- Warum sollten Bebauungspläne mit GIS erzeugt werden?
- Welche Bibliotheken gibt es um eine Exportschnittstelle für XPlanGML-Dateien zu realisieren?
- Wie können diese Bibliotheken weiterentwickelt werden, um diese für die Erzeugung von GML-Daten zu nutzen?
- Wie kann mit dem Stand der Technik der Export nach XPlanGML realisiert werden?
- Wie kann die Exportschnittstelle von anderen Entwicklern im Anschluss dieser Arbeit weiterentwickelt werden?
- Welchen Mehrwert erhalten die Kommunen durch den Export nach XPlanGML?
- Warum ist das Dateiformat XPlanGML im Planungsprozess so wichtig?

## 1.2 Struktur der Arbeit

Für den Einstieg in diese Arbeit werden in Kapitel 2 die fachlichen Grundlagen erläutert. Dabei wird zuerst auf die Digitalisierung von Bebauungsplänen eingegangen, wobei die rechtlichen Rahmenbedingungen sowie der deutsche Standard XPlanung für die Realisierung beleuchtet werden. Anschließend wird die Umsetzung von Bebauungsplänen mit GIS betrachtet. Dabei werden als Erstes Geoinformationssysteme definiert, mit denen Bebauungspläne erzeugt werden können. Danach werden die Open Source GIS-Software QGIS und das enthaltene Plugin XPlanung vorgestellt, mit dem in dieser Arbeit die Bebauungspläne erzeugt werden sollen.

Im darauffolgenden Kapitel 3 wird das Material und die Methodik vorgestellt. Dabei wird im ersten Teil des Kapitels die im Rahmen der Machbarkeitsanalyse für den Export untersuchte Software und Bibliotheken vorgestellt. Anschließend wird im zweiten Teil des Kapitels 3 die Methodik und Durchführung der Arbeit beschrieben. Bei der Entwicklung muss zuvor auf die Struktur des vorhandenen Plugins eingegangen werden, da das Export-Tool in diese Umgebung integriert werden muss. Des Weiteren müssen bei der Entwicklung auch einige von dem Standard XPlanung vorgegebenen Konformitätsbedingungen bei der Entwicklung eingehalten werden. Nachdem diese Grundlagen für die Programmierung der Exportschnittstelle untersucht wurden, wird die prototypische Entwicklung des Exports nach XPlanGML dargelegt. Hierzu wird der Programmablauf mit dazugehörigen Codeausschnitten

vorgelegt und abschließend ein Schema geliefert, mit dem das Export-Tool realisiert werden kann.

Danach wird analysiert, was mit der Exportschnittstelle für das Datenformat XPlanGML erreicht wurde. Daraufgehend findet eine kritische Diskussion der Ergebnisse statt und abschließend werden ein Fazit gezogen sowie Ausblick dargestellt.

## 2 Fachliche Grundlagen

In diesem Kapitel werden die fachlichen Grundlagen für diese Arbeit geschaffen. Dabei wird im ersten Abschnitt die Digitalisierung von Bebauungsplänen vorgestellt. Dazu werden zum einen die rechtlichen Grundlagen von Bebauungsplänen sowie der deutsche Standard XPlanung dargestellt. Anschließend wird im zweiten Teil auf die praktische Umsetzung von Bebauungsplänen eingegangen. Hierfür werden zuerst Geoinformationssysteme definiert und anschließend die Open Source Software QGIS mit dem dort enthaltenen Plugin des Standards XPlanung für die Umsetzung präsentiert.

### 2.1 Digitalisierung von Bebauungsplänen

Um die Digitalisierung von Bebauungsplänen mit dem deutschen Standard XPlanung zu durchdringen, müssen als erstes die gesetzlichen Grundlagen von Bebauungsplänen in Betracht gezogen werden. Diese werden in Deutschland mit dem Baugesetzbuch (BauGB) definiert. Danach wird der deutsche Standard XPlanung für die Umsetzung von Bebauungsplänen vorgestellt. Mit diesem Standard sollen in Deutschland nicht nur die Bebauungspläne, sondern auch die Flächennutzungspläne, Raumordnungspläne, Landschaftspläne sowie sonstige räumliche Planwerke umgesetzt werden (KRAUSE 2022b).

#### 2.1.1 Bebauungspläne

Bebauungspläne fallen nach BauGB unter die Bauleitplanung. Nach §1 Absatz 2 BauGB wird in der Bauleitplanung zwischen der vorbereitenden Bauleitplanung (Flächennutzungsplan) und der verbindlichen Bauleitplanung (Bebauungsplan) unterschieden. Mit diesen Plänen wird zum einen die Vorbereitung (Flächennutzungsplan) und zum anderen die Leitung (Bebauungsplan) der baulichen und sonstigen Nutzung der Grundstücke realisiert (§ 1 Absatz 1 BauGB). Diese Pläne müssen von den Gemeinden aufgestellt werden, wenn diese aufgrund städtebaulicher Entwicklungen erforderlich werden (§ 1 Absatz 3 Satz 1 BauGB). Mit den Bauleitplänen soll eine nachhaltige Entwicklung im Städtebau erreicht werden (§1 Absatz 5 Satz 1 BauGB).

Die rechtlichen Rahmenbedingungen von Bebauungsplänen sind im BauGB § 8 - § 10 der verbindlichen Bauleitplanung festgehalten. BauGB § 8 Abs. 1 Satz 1 besagt, dass mit dem Bebauungsplan die rechtsverbindlichen Festsetzungen für die städtebauliche Entwicklung festgelegt werden. Daher ist die Grundlage jeder baulichen Maßnahme der Bebauungsplan

und diese muss sich dabei an das Baugesetzbuch halten. Nach § 10 Abs. 3 Satz 2 BauGB müssen Bebauungspläne für jedermann zur Einsicht bereitgestellt werden. Das bedeutet, dass sie mittels entsprechender Dienste oder auf GIS-Server publiziert werden müssen (STRÖBL 2016).

Damit die Pläne bundesweit einheitlich sind, wird durch § 2 Abs. 5 Nr. 4 BauGB die Planzeichenverordnung (PlanZV) erlassen. Mit dieser Verordnung werden die Ausarbeitung der Bauleitpläne und die Darstellung des Planinhalts geregelt. Nach § 2 Abs. 1 Satz 1 PlanZV müssen die Bauleitpläne nach den Planzeichen in den Anlagen der Verordnung umgesetzt werden. Wichtig dabei ist, dass die verwendeten Planzeichen in den Bauleitplan erläutert werden (§ 2 Abs. 4 PlanZV).

### 2.1.2 XPlanung

Bei den Bau- und Planungsprozessen in den öffentlichen Verwaltungen werden heutzutage immer mehr unterschiedliche IT-Anwendungen und -Systeme verwendet. Die Produkte sollen diese Bau- und Planungsprozesse eigentlich vereinfachen und dabei gewinnbringend eingesetzt werden jedoch müssen in vielen Fällen beim Austausch von Daten zwischen Prozessbeteiligten ebendiese Daten manuell erfasst werden oder können sogar abhandenkommen (KRAUSE ET AL. 2016, S. 679). Damit dieser Entwicklung entgegengewirkt werden kann, sollte an den Gemeinden sowie für die Projektbeteiligten Standards in den Planungsprozessen eingeführt werden.

Im Rahmen der eGovernment Initiativen von Deutschland-Online und Media@Komm-Transfer wurden Datenmodelle und Datenformate definiert, mit denen geometrische und semantische Inhalte von Planwerken der Bauleit- und Landschaftsplanung sowie der Raumordnung erzeugt werden können. Dieser deutsche Standard für die Umsetzung der Planungsprozesse wird XPlanung genannt (BENNER ET AL. 2005, S. 487). Zusätzlich dazu gibt es den Standard XBau für die alphanumerischen Inhalte von Bauanträgen. In beiden Standards steht das „X“ für XML (Extensible Markup Language); sie reihen sich in die XML-Formate von der Initiative XML der öffentlichen Verwaltung (XÖV) ein (KRAUSE ET AL. 2016, S. 679).

An dem Projekt XPlanung waren, unter der Führung der Freien und Hansestadt Hamburg und des Landkreises Bad Segeberg, verschiedene Planungs- und Vermessungsämter, Kommunen, Spezialisten für kommunale Datenverarbeitung, Firmen sowie Vertreter\*innen aus der Forschung beteiligt (BENNER UND KRAUSE 2006, S. 33). Dieser Standard XPlanung mit dem

objektorientierten Datenaustauschformat XPlanGML wurde im Jahr 2017 vom IT-Planungsrat verbindlich eingeführt (KRAUSE 2022b). Mit der Einführung dieses Standards muss dieser von allen Kommunen in Deutschland innerhalb von fünf Jahren umgesetzt werden (IT-PLANUNGSRAT 2017).

### **Ziele und Vorteile**

Mit der Standardisierung der Datenmodelle und Datenformate soll die Aufstellung, Veröffentlichung und interoperable Nutzung von den Planwerken realisiert werden. Diese ist in Abbildung 1 mit dem Schaubild dargestellt, das darlegt,

warum über einen Standard die Planungsbeteiligten besser in den Planungsprozess miteinbezogen werden können. Somit können die Planungsbeteiligten über verschiedene IT-Systeme auf die Planungsdaten mit dem Datenaustauschstandard zugreifen und arbeiten, ohne dass es zu einem Verlust von Daten kommt. Dafür muss das Datenaustauschformat für die Beteiligten zentral auf einem Server zur Verfügung gestellt werden (WÜRRIEHAUSEN UND MÜLLER 2012, S. 736 ff). Hinzu kommt, dass mit dem Standard auch alle bestehenden gesetzlichen Grundlagen von dem BauGB, der Baunutzungsverordnung (BauNVO) und der PlanZV umgesetzt werden (BILL 2010, S. 666). Neben der Interoperabilität des Standards, soll mit XPlanung eine optimale Nutzung der vorhandenen Ressourcen sowie die Beschleunigung und Vereinfachung des Prozesses verfolgt werden. Des Weiteren wird mit den Zielen des Standards die Kostenreduktion der Planungsverfahren verfolgt (WÜRRIEHAUSEN UND MÜLLER 2012, S. 736 ff).

Nicht nur die öffentliche Verwaltung profitiert von dem Standard, sondern gerade auch die am Planungsprozess beteiligten Akteure. Hierbei sind neben den Kommunen z. B. auch Planungsbüros, Bürger\*innen, Wirtschaft, Träger öffentlicher Belange, Nachbarkommunen und übergeordnete Gebietseinheiten inkludiert. Die Vorteile für die Beteiligten sind zum Beispiel, dass die Daten aus der Planung vereinfacht übernommen werden können und bei weiterer Bearbeitung eine Versionierung erzeugt werden kann. Über entsprechende standardbasierte Plattformen können sich die verschiedenen beteiligten Akteure in den Planungsprozess einbringen. Für interessierte Bauwillige können im Internet WMS-Dienste oder Server bereitgestellt werden, wodurch einfach und unverbindlich Information zu den Plänen bezogen werden können. Ebenfalls werden benachbarte Kommunen sowie

übergeordnete Ämter frühzeitig in die Planung einbezogen, sodass Fehler und Konflikte auf den verschiedenen Ebenen vermieden werden können. (KRAUSE 2022b)



Abbildung 1: Zusammenarbeit bei dem Planungsprozess ohne (links) und mit (rechts) dem XPlanung-Datenaustauschstandard. (KRAUSE 2022b)

## Struktur des Standards

Um die Struktur des komplexen Standards zu überblicken, ist in Abbildung 2 eine vereinfachte Struktur des Standards zu sehen. Die gesamte Struktur des Standards ist durch die UML-Diagramme oder durch den Objektartenkatalog der jeweiligen Version des Standards nachvollziehbar. In der Struktur ist es so, dass es Oberklassen gibt, vor denen „XP“ steht; aus ihnen werden die Klassen zu den einzelnen Planwerken abgeleitet. Bei den Planwerken unterscheidet man dann die Klassen zwischen Bebauungsplan (BP), Flächennutzungsplan (FP), Regionalplan und landesweiten Raumordnungsplan (RP), Landschaftsplan (LP) sowie sonstige raumbezogene Planwerke (SO). (BENNER 2020b, S. 7)

Neben dieser Unterteilung in den Klassen der Planwerke gibt es in dem Standard eine weitere hierarchische Struktur für die Klassen. An oberster Stelle dieser Hierarchie ist die Plan-Klasse, mit der das gesamte Plan-Gebiet definiert wird. Die Oberklasse für die Plangebiete heißt „XP\_Plan“ und spezifiziert sich dann mit den entsprechenden Kürzeln zu den Unterklassen zu den Planwerken wie z. B. für die Bebauungspläne „BP\_Plan“. Die jeweiligen Pläne zu den Planwerken werden in der Bereichsebene erstellt. Hierbei können ein oder mehrere Bereiche in einem Plan-Gebiet liegen. Auch die Bereiche haben eine abstrakte Oberklasse („XP\_Bereich“) und die abgeleiteten Klassen zu den verschiedenen Planwerken, wie für die Bebauungspläne „BP\_Bereich“. Um die zugehörigen Planinhalte in den entsprechenden Plan-Bereichen darzustellen, gibt es zwei Möglichkeiten. Zum einen ist es möglich, den Karteninhalt

für den Bereich als Rasterdarstellung mit einer oder mehreren Rasterdarstellungen abzubilden. Dies kann verwendet werden, wenn die Pläne analog vorliegen und als Rasterkarte in den zugehörigen Bereich georeferenziert werden. Zum anderen können die Karteninhalte über Vektordarstellungen in den Bereichen präsentiert werden. Dafür werden von dem Standard für die verschiedenen Planwerke adäquate vektorielle Plan-Objekte angeboten. Auch für die Objekte beginnt die Klassen-Hierarchie mit der abstrakten Oberklasse „XP\_Objekt“. Für die Planwerke gibt es ebenfalls nochmal Oberklassen mit dem entsprechenden Kürzel zu den Planwerken, wie z. B. „BP\_Objekt“. Aus diesen Objekten zu den Planwerken werden die Objekte zu den Planinhalten abgeleitet. Zusätzlich erhalten die Objekte der Pläne eine Klasse für die textlichen Planinhalte. Mit dieser Klasse werden die Planinhalte dargelegt, die nicht formalisiert sind, sondern als freier Text vorliegen. Auch hier gibt es wieder eine abstrakte Oberklasse „XP\_TextAbschnitt“ und zu den Planwerken eine spezifizierte Klasse, wie bei den Bebauungsplänen der „BP\_TextAbschnitt“. (BENNER 2020b, S. 7 ff)

Somit ist schon aus der vereinfachten Struktur in Abbildung 2 des XPlanung-Datenmodell ersichtlich, wie komplex die Daten des Datenaustauschformats XPlanGML werden können. Dieses Dateiformat wird im nächsten Abschnitt vorgestellt.



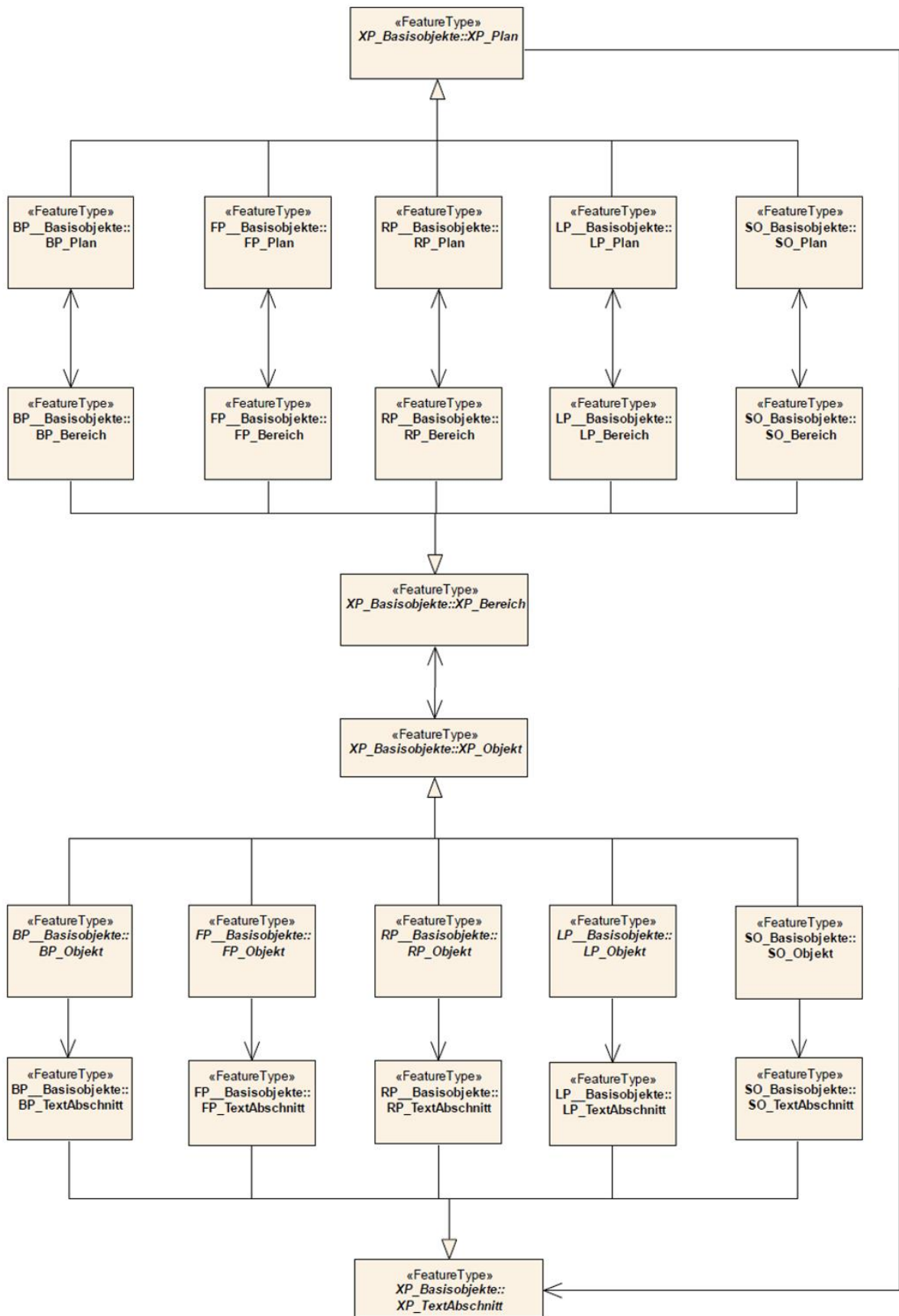


Abbildung 2: Vereinfachte Struktur des XPlanung-Datenmodell. (BENNER 2020b, S. 8)

## **XPlanGML-Datenaustauschformat**

XPlanGML ist das objektorientierte Datenaustauschformat von XPlanung, welches mit dem internationalen OGC Standard GML-3 (Geography Markup Language) kompatibel ist (BENNER ET AL. 2008, S. 244). Für den Aufbau dieses Datenaustausch-Format wurden von der XLeitstelle verschiedene Versionen veröffentlicht, wobei die aktuellste Version die XPlanung Version 6.0 ist (KRAUSE 2022b). Die Struktur des Datenaustauschformats wird in XML Schema Definition Dateien, den so genannten XSD-Dateien, gespeichert (DUAN 2022). Wie in den XML-Schemata des Standards XPlanung steht, wurden diese aus den UML-Klassendiagrammen mit der UmlToXmlTransformation-Software des Karlsruher Institut für Technologie (KIT) automatisch generiert, die ebenfalls bei dem ALKIS-Basischema verwendet wurde. Somit ist das erstellte XML-Schema ebenfalls NAS-konform (Normbasierte Austauschchnittstelle), ohne direkt auf das ALKIS-Schema zuzugreifen. (BENNER ET AL. 2008, S. 244).

Die XPlanGML-Datei muss neben der syntaktischen Umsetzung nach dem XML-Schema, zusätzlich auch geometrische und semantische Regeln befolgen (DUAN 2022). Das bedeutet, dass die Geometrien und Attribute in der XPlanGML-Datei in der XPlanung-Version 5.2.1 die Regeln aus dem Dokument „Konformitätsbedingungen XPlanung 5.2.1“, die von BENNER (2020a) aufgestellt wurden, befolgen müssen. Somit muss die XPlanGML-Datei nicht nur nach dem XML-Schema valide sein, sondern auch die speziellen Konformitätsregeln erfüllen. Diese speziellen Bedingungen lassen sich in globale und klassenspezifische Regeln unterteilen. Bei den globalen Bedingungen gibt es Regeln für die Einschränkung des GML-Standards, allgemeine geometrische und topologische Bedingungen, Bedingungen für die Bezeichnung von Maßeinheiten und Koordinaten-Referenzsystemen sowie Vorschriften für die geometrische Erfassung von raumbezogenen Planinhalten. Damit ist zu sehen, dass die XPlanGML-Dateien weiter spezifiziert werden gegenüber dem OGC Standard GML-3. Bei den klassenspezifischen Regeln handelt es sich um Einschränkungen, die sich auf Oberklassen und Klassen der Planwerke beziehen. Die Regeln, die für die Oberklassen gelten, werden an deren Unterklassen weitervererbt. Das Ziel dieser Konformitätsbedingungen ist es, die Qualität der Pläne von XPlanung zu steigern und die Auswertung für die XPlanung-Fachsysteme zu vereinfachen. Somit müssen sich gerade die Softwareentwickler in den Fachsystemen nach diesen Spezifikationen richten. Ein wichtiger Punkt ist hier die Exportschnittstelle, die bei dem Export sowohl die syntaktische Validität nach dem XML-Schema als auch die

Konformitätsregeln realisieren muss (BENNER 2020a, S. 14). Ob eine XPlanGML-Datei valide ist, kann mit dem Validator von KRAUSE (2022a) mit dem Namen „XPlanValidator“ geprüft werden.

## 2.2 Umsetzung von Bebauungsplänen mit GIS

Bei der Umsetzung von Bebauungsplänen werden hauptsächlich Computer Aided Design (CAD)- oder GIS-Software verwendet. Gerade bei den GI-Systemen gibt es einige Vorteile gegenüber den CAD-Systemen, da der Raumbezug in dieser Software berücksichtigt wird. Damit können bei den verschiedenen Planungsschritten die räumlichen Bezüge berücksichtigt und analysiert werden. Da kommerzielle GIS-Software verhältnismäßig teuer ist, müssen gerade bei Gemeinden mit kleinem Budget die Open Source Produkte aus dem GIS-Bereich in Betracht gezogen werden. Die bekannteste Open Source Software im GIS-Bereich ist QGIS, in der schon ein Plugin mit dem Standard XPlanung realisiert wurde. Sowohl die Software QGIS als auch das Plugin XPlanung werden nachfolgend näher vorgestellt. Begonnen wird in diesem Abschnitt aber mit der Begriffsdefinition zu Geoinformationssystemen.

### 2.2.1 Geoinformationssystem (GIS)

Mit dem Begriff Geoinformationssystem wird ein System beschrieben, das aus Hardware Software, Daten und Anwendungen besteht. Mit diesem System kann die Erfassung, Verwaltung, Analyse und Präsentation (EVAP-Modell) von raumbezogenen Raster- und Vektordaten durchgeführt werden (DE LANGE 2020, S. 375). Die Bereiche für die Anwendungen von einem GIS sind sehr breit gefächert, wie zum Beispiel der Katasterbereich, Verkehr, Geomarketing, Landwirtschaft sowie dem Planungsbereich u.v.m. (BARTELME 2005, S. 33 ff).

Da die Bebauungspläne in den Planungsbereich fallen, wird dieser Bereich in der Arbeit näher betrachtet. Für den Planungsbereich ist es sinnvoll im GIS zu arbeiten, da hier sowohl kleinräumige, regionale, als auch nationale und internationale Planungsmaßnahmen durchgeführt werden. Bei diesen Planungen muss meist über das Planungsgebiet hinausgeschaut werden, da die Auswirkungen auf das anthropogene Umfeld und Ökosystem analysiert und berücksichtigt werden (BILL 2010, S. 11). Bei der kommunalen Bauleitplanung ist zu beobachten, dass immer mehr GIS-Fachschalen zur Aufstellung der Pläne in den Kommunen genutzt werden, wobei sich der Standard XPlanung hierfür als Grundlage entwickelt hat (BILL 2010, S. 673).

Bei den GIS-Produkten wird zwischen Open Source und kommerzieller Software unterschieden und in den beiden Lagern gibt es verschiedene Software-Produkte. Der Marktführer im kommerziellen Bereich ist ArcGIS und im Open Source Bereich QGIS (DE LANGE 2020, S. 2). In dieser Arbeit wird lediglich das Open Source Programm QGIS verwendet, da gerade dieses günstig von Kommunen mit geringerem Budget genutzt werden kann. Diese Software wird im nächsten Kapitel näher erläutert.

### 2.2.2 QGIS

Wie bereits erwähnt ist QGIS ein freies, geographisches Informationssystem, welches unter der GPL-Lizenz (General Public License) steht (QGIS 2022c). Es soll gerade gegenüber den teuren und proprietären GIS-Produkten eine Alternative darstellen, die kostenfrei genutzt werden kann (QGIS 2022a). Die Software wird von den folgenden Betriebssysteme unterstützt: Linux, Unix, Mac OSX, Windows und Android (QGIS 2022c). Die Software hält alle allgemeinen Funktionen und Merkmale eines GIS vor. Damit es ein vollständiges GIS ist, wurden sowohl die Möglichkeit eines GIS-Viewers realisiert als auch die Datenerfassung, fortgeschritten GIS-Analysen sowie die Präsentation in verschiedenen Formen. Für die Arbeit mit verschiedenen Dateiformaten ist es möglich, verschiedene Raster- und Vektordaten einzuladen sowie die Verbindung mit Geodatenbanken aufzubauen. (QGIS 2022a).

Zusätzlich bietet QGIS die Möglichkeit die Software mit verschiedenen Plugins zu erweitern. Hierbei liegen schon über 1.600 Plugins vor, die von unabhängigen Organisationen und Entwickler\*innen zur Verfügung gestellt werden (QGIS 2022b). Aber auch die eigene Erstellung eines Plugins ist in QGIS möglich. Ein Plugin kann mit Hilfe der Plugins Plugin Builder, Plugin-Reloader sowie der Software Qt Designer erzeugt werden. Mit dem Plugin Builder können Vorlagen für ein Plugin mit den benötigten Dateien erzeugt werden und der Plugin-Reloader ist für die Aktualisierung des weiterentwickelten Programmcodes zuständig. Mit der Software Qt Creator wird für den Bau von Graphical User Interface (GUI) ein Werkzeug zur Verfügung gestellt. (UJAVAL GANDHI 2021). Der Qt Creator arbeitet nach der Methode what-you-see-is-what-you-get (WYSIWYG) und somit kann die graphische Benutzeroberfläche mit Vorlagen von Button, Fenstern und Dialogen zusammengestellt werden (QT COMPANY 2022).

### 2.2.3 Plugin XPlanung

Für der Standard XPlanung ist in der Software QGIS bereits ein Plugin vorhanden. Dieses Plugin wurde von STRÖBL (2021a) auf GitHub veröffentlicht und kann unter dem Erweiterungsmanager von QGIS heruntergeladen werden. Bei dem Plugin ist zu beachten, dass dieses lediglich bis in die XPlanung Version 5.2 realisiert wurde (STRÖBL 2021a). Damit das Plugin einsatzbereit ist, müssen zusätzlich noch das Plugin „Data-Driven Input Mask“ heruntergeladen werden und eine PostgreSQL/PostGIS-Datenbank erzeugt werden, die mit dem XPlanung-Plugin verbunden wird (STRÖBL 2018).

Mit dem Plugin „Data-Driven Input Mask“ können die Sachdaten für die XPlanung-Objekte eingegeben werden (STRÖBL 2020). Es werden automatisiert aus den PostgreSQL-Layer Eingabemasken erzeugt. Zusätzlich werden auch alle Datenbankbeschränkungen mit dieser Maske berücksichtigt. Ein Beispiel für eine Eingabemaske für das XPlanung-Objekt „BP\_StrassenVerkehrsFlaeche“ ist in der Abbildung 3 gegeben. Des Weiteren wird noch eine Suchmaske von dem Plugin angeboten, mit dem über die Attribute die entsprechende XPlanung-Objekte gesucht werden können (STRÖBL 2019).

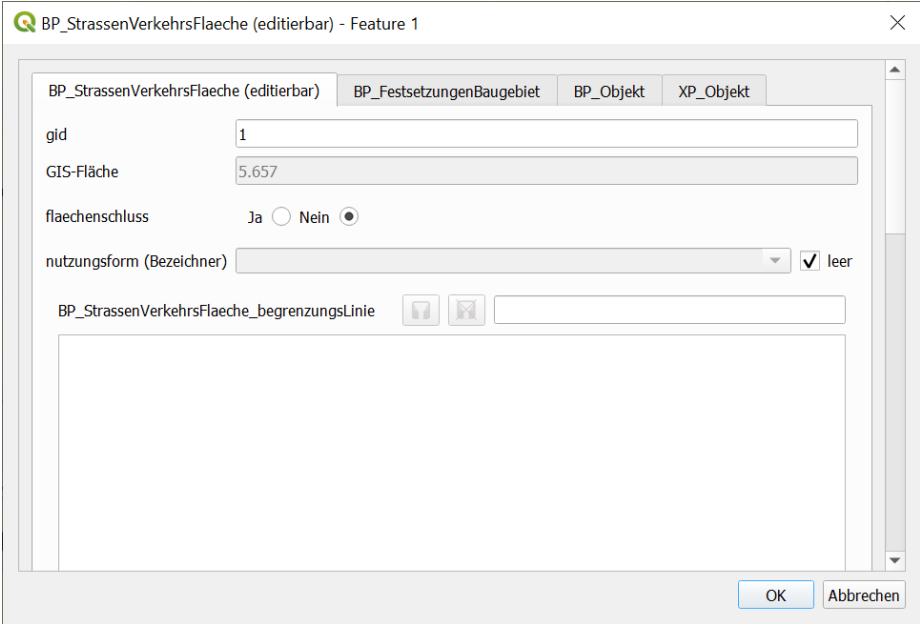


Abbildung 3: Beispiel für eine Data-Driven Input Mask für das XPlanung-Plugin. (eigene)

Für die Einrichtung der PostgreSQL/PostGIS-Datenbank werden von STRÖBL (2021b) auf GitHub SQL-Skripte zur Verfügung gestellt, mit denen die Erzeugung der Datenbank durchgeführt werden kann. Hierbei ist es wichtig, dass die Skripte nach der vorgegebenen Reihenfolge

ausgeführt werden, wie sie in der Vorbereitung und Durchführung der README.md-Datei angegeben sind. (STRÖBL 2021b)

Nach dem Erzeugen und Verbinden der Datenbank sowie dem Herunterladen des Plugins „Data-Driven Input Mask“ ist das Plugin XPlanung einsatzbereit. Anschließend können die verschiedenen Objektarten von den Planwerken eingeladen werden. Für das Erzeugen der XPlanung-Objekte können die Digitalisierungswerkzeuge von QGIS genutzt werden. Im ersten Schritt müssen immer für jedes Planwerk ein Plan-Gebiet und anschließend ein zugeordnetes Bereich-Gebiet zu einem Plan-Gebiet erzeugt werden. Für die Erzeugung der Objekte des Plans muss immer ein aktiver Bereich angegeben sein, dem die Objekte zugeordnet werden. Für Bebauungspläne gibt es die Möglichkeit, eine automatische Erzeugung von Nutzungsschablonen zu integrieren (STRÖBL 2018). Des Weiteren können auch eigene Stile zur Gestaltung der Pläne erzeugt und gespeichert werden. Für den Import von XPlanGML-Dateien ist ebenfalls ein Tool in dem Plugin vorhanden (STRÖBL 2020).

Das Plugin ist zwar schon einsatzfähig, um Pläne nach dem Standard zu erzeugen, aber es fehlen trotzdem noch einige Funktionen und Bestandteile. STRÖBL (2020) hat dafür die folgende ToDo-Liste mit fehlenden Punkten aufgestellt:

- Export nach XPlanGML,
- Standardstile und svg-Grafiken für diverse Objektarten,
- RasterplanBasis- und RasterplanAenderung-Klassen,
- Anlegen und Verwalten eine XPlan-Datenbank aus QGIS heraus.

Diese Liste ist keine vollständige Liste von Tools, die in dem Plugin fehlen, sondern lediglich Punkte die als nächstes umgesetzt werden sollten. Die Reihenfolge in der Liste gibt keine Priorisierung an. (STRÖBL 2020)

In der weiteren Arbeit steht der Export nach XPlanGML im Vordergrund. Dabei wird in dem nächsten Abschnitt das Material und die Methodik für die Machbarkeitsanalyse für die Exportschnittstelle vorgestellt.

## 3 Material und Methodik

Um den Umfang der Arbeit nicht zu überschreiten, wird der Fokus auf einen Punkt der ToDo-Liste gelegt. Da der Export von XPlanGML-Dateien ein immens wichtiges Tool ist, wird die Umsetzung einer solchen Exportschnittstelle in dieser Arbeit betrachtet. Im Folgenden wird die Methodik für den Export nach XPlanGML dargestellt. Hierbei wird zu Beginn dieses Kapitels die untersuchte Software und Bibliothek vorgestellt, mit denen die Machbarkeit einer Umsetzung des Exports analysiert wurde. Im zweiten Teil des Kapitels wird die Methodik und Durchführung der XPlanGML-Exportschnittstelle dargelegt. Dabei wird zuerst der schon vorliegende Stand des Plugins betrachtet. Danach werden noch einige Anforderungen an die Softwareentwicklung über Konformitätsbedingungen gestellt und abschließend wird die prototypische Entwicklung der XPlanGML-Exportschnittstelle dargestellt.

### 3.1 Software und Bibliotheken

In diesem Unterkapitel werden die betrachtete Software und Bibliotheken für die Machbarkeitsanalyse der Exportschnittstelle nach XPlanGML vorgestellt, beginnend mit der empfohlenen Software für den Export für die Daten aus der PostgreSQL-Datenbank von QGIS. Anschließend werden die Bibliotheken aufgezeigt, mit denen ein Export umgesetzt werden könnte. Dabei werden lediglich Bibliotheken untersucht, die in Python verwendet werden können, da der vorhandene Programmcode des Plugins in Python umgesetzt wurde und somit mit den Bibliotheken auf diesen aufbauen muss.

#### 3.1.1 Hale Studio

Nach STRÖBL (2020) wird für den Export von XPlanGML in QGIS die Software Hale Studio empfohlen, da in dem QGIS-Plugin der Export noch nicht vorliegt. Hale Studio ist eine Open Source Software der Firma wetransform, die eine Ausgründung aus dem Fraunhofer Institut für geographische Datenverarbeitung ist (WETRANSFORM GMBH 2020).

Mit dieser Software soll ein schneller und interaktiver Datenaustausch von komplexen Daten geschaffen werden. Sie schafft die Verbindung zu vielen Standards. Bei der Software ist es in drei Schritten machbar, die Transformation durchzuführen. Im ersten Schritt muss das Quellschema angegeben. Danach kann im zweiten Schritt das Zielschema gewählt werden und im letzten Schritt müssen nur noch die Transformationsfunktionen sowie die erforderlichen

Parameter gesetzt werden. Bei der Transformation kann mit vielen vorhandenen offenen Standards gearbeitet werden, wie OGC GML und CityGML, INSPIRE, ALKIS/NAS, IFC oder jeden anderen XML- oder JSON-basierten Standard. Zusätzlich wird auch die Arbeit auf Datenbanken mit PostgreSQL, Oracle, File-Geodatabases und anderen Formaten unterstützt. (WETRANSFORM GMBH 2022a)

In Abbildung 4 ist die Benutzeroberfläche von Hale Studio zu sehen. Hierbei sieht man den Versuch, die Datentransformation nach XPlanGML mit den Quelldaten aus der Datenbank von dem Plugin XPlanung von QGIS durchzuführen. Hierbei wurde in der Software als Quellschema (Source) die PostgreSQL-Datenbank und als Zielschema (Target) das XPlanGML-Schema angegeben. Bei der Transformationsfunktion und dem Verbinden der erforderlichen Parameter kam es zu Schwierigkeiten, da dies nicht intuitiv durchgeführt werden konnte. Hierzu gab es sowohl im Support Forum als auch im User Guide der Software keine Anleitung zur Durchführung. Somit ist es an dem Schritt gescheitert, in der Software eine XPlanGML-Datei zu exportieren.

Um die komplexen Strukturen von der PostgreSQL-Datenbank mit dem XPlanGML-Format zu verbinden, wird von WETRANSFORM GMBH (2022b) eine Fortbildung in Höhe von 800 € angeboten.

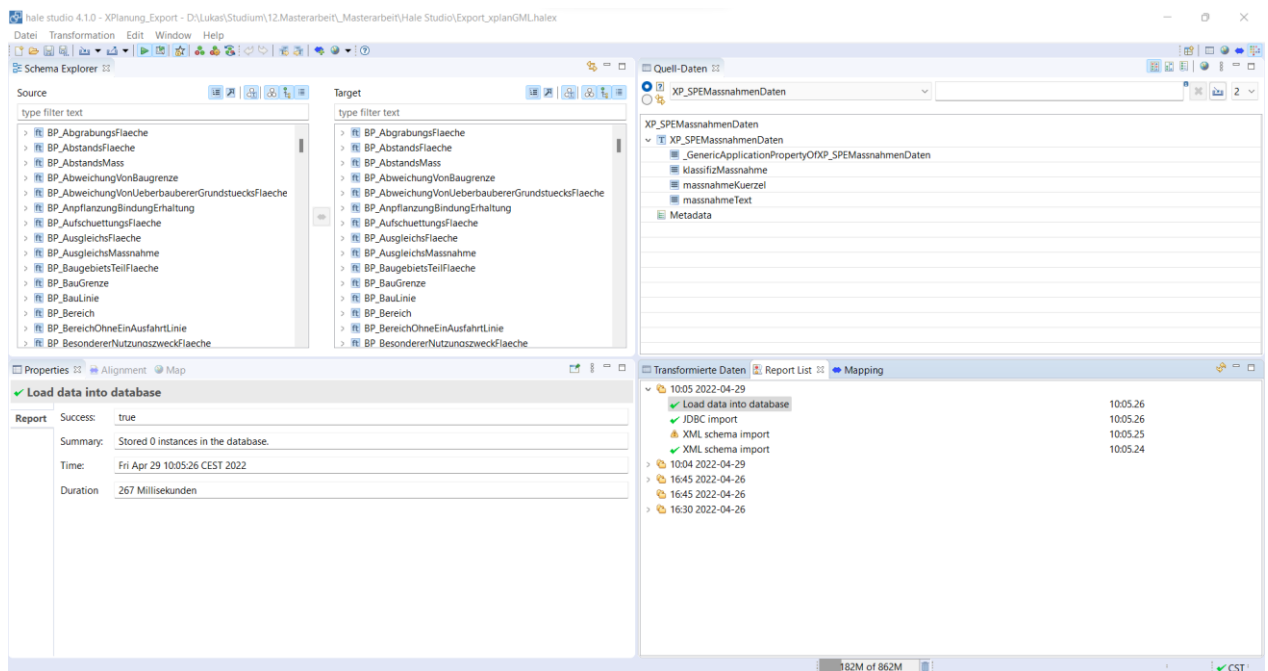


Abbildung 4: Benutzeroberfläche von hale Studio mit dem Versuch des Exports von XPlanGML-Dateien. (eigene)



Somit wird deutlich, dass die Open Source Software zwar gratis zur Verfügung gestellt wird, aber so komplex ist, dass lediglich mit einer Schulung der Export nach XPlanGML umsetzbar ist. Der jedoch wichtigste Punkt, der gegen die Software spricht, ist, dass eine Umsetzung des Exports in dem QGIS-Plugin XPlanung entwickelt werden soll und keine ausstehende Software hinzugezogen werden sollte.

### 3.1.2 GDAL/OGR-Bibliothek

Die Open Source GDAL/OGR-Bibliothek ist die bekannteste Bibliothek für den Zugriff auf und der Verarbeitung von räumlichen Daten. Es gibt mit dieser Bibliothek die Möglichkeit, verschiedene Raster- und Vektorformate zu lesen und weiter zu verarbeiten. Dabei werden mit der Geospatial Data Abstraction Library (GDAL) über 140 verschiedene Rasterdatenformate und dem OGR, die der OGC Simple Features Spezifikation folgt, über 80 Vektordatenformate unterstützt. Der Zugriff auf die Bibliotheken kann über Python, Java und viele weitere Programmiersprachen realisiert werden. (OSGEO LIVE 2021)

Zu den verschiedenen Datenformaten, die von der Bibliothek vorgehalten werden, gibt es Programme, die diese Datenformate weiterverarbeiten. Diese Programme unterteilen sich in Raster-Programme, Multidimensionale Raster-Programme, Vektor-Programme und Geografische Netzwerk-Programme. Für diese Arbeit wird nur das Programm ogr2ogr von den Vektorprogrammen in Betracht gezogen. (WARMERDAM UND ROUAULT 2022, S. 45 ff)

#### **Vektor-Programm ogr2ogr**

Mit Hilfe von diesem Programm kann die Konvertierung von einem Vektordatenformaten zum anderen Vektordatenformat durchgeführt werden. Während des Prozesses der Konvertierung können verschiedene Operationen durchgeführt werden, wie z. B. räumliche oder attributive Selektionen, eine Reduzierung der Datenmenge, das Transformieren des Koordinatenreferenzsystem sowie viele weitere Operationen. (WARMERDAM UND ROUAULT 2022, S. 138 ff)

Da bei dem Export der XPlanGML eine GML-Datei mit einem entsprechenden XML-Schema verwendet werden soll, muss das OGR Datenformat Geography Markup Language driven by application schemas (GMLAS) bei der ogr2ogr Funktion angewandt werden. Somit muss eine Transformation der Daten von PostgreSQL nach GMLAS durchgeführt werden. (WARMERDAM UND ROUAULT 2022, S. 635 ff)

Für die Erstellung von schema-gesteuerten GML-Dokumenten wird mit dem GMLAS-Treiber eine Konvertierung von Quelldatensätzen zur Verfügung gestellt. Hierbei ist der typische Arbeitsablauf, dass die Daten über die Lese-Funktion des GMLAS-Treibers ausgelesen und in eine SQLite/Spatialite/PostGIS Datenbank geschrieben werden. Dabei werden die importierten Features möglicherweise angepasst und abschließend die Datenbank als GML-Dokument exportiert. Für das Erzeugen des Exports muss immer das XML-Schema zur Verfügung stehen, aber von dem Programm wird nicht garantiert, dass die erzeugte Datei gegenüber dem Schema valide ist. Bei der Konvertierung aus einer Quelldatenbank gibt es auf Grund des Arbeitsablaufs des Programms einige Einschränkungen in den Optionen der Transformation. Es können lediglich räumliche und attributive Filterungen auf den obersten Ebenen angewandt werden. Die Verknüpfungen zu den unteren Ebenen der Daten sind von dem Filter nicht betroffen. (WARMERDAM UND ROUAULT 2022, S. 635 ff)

Durch die Einschränkungen bei der Datenbankabfrage ist auch diese Funktion der OGR-Bibliothek für den Export von XPlanGML-Dateien nicht anwendbar. Es ist nur möglich, gesamte Tabellen abzufragen; in den abgefragten Tabellen können aber verschiedene Teile von den XPlanung-Objekten vorliegen, die nicht den abgefragten Gebieten zugehören.

### 3.1.3 lxml-Bibliothek

Die lxml-Bibliothek ist eine Open Source Bibliothek, mit der HTML- und XML-Daten verarbeitet und erzeugt werden können. Mit dem XML-Toolkit wird eine Python-Schnittstelle für die beiden C-Bibliotheken libxml2 und libxslt zur Verfügung gestellt (RICHTER 2022). Die beiden C-Bibliotheken wurden im Rahmen des GNOME-Projektes für das Lesen und die Transformation von XML-Dateien entwickelt (WELLNHOFER 2022a, 2022b). Mit der Bibliothek soll auch ElementTree-API erweitert werden mit dem Modul lxml.etree (RICHTER 2022). Die ElementTree-API ist eine einfach API zum Verarbeiten und Erstellen von XML-Dateien (PYTHON SOFTWARE FOUNDATION 2022d). Mit lxml.etree werden viel mehr Funktionalitäten gegenüber der ElementTree-API angeboten, wie bspw. die Unterstützung von XPath, XLT und XML-Schema (RICHTER 2022).

Um eine XML-Baumstruktur erzeugen zu können, muss ein ElementTree erzeugt werden. In diesen ElementTree ist es möglich, XML-Elemente als Objektcontainer anzulegen. Um in den Objektcontainer untergeordnete Elemente anzulegen, können in diesen Elementen

sogenannte SubElemente angelegt werden. Für die Elemente und SubElemente können ebenfalls Attribute und Werte angelegt werden. (RICHTER 2022)

Mit dem ElementTree, den Elementen und SubElementen ist es sehr einfach und effizient, XML-Daten zu erzeugen. Dies ist in dem nachfolgenden eigenen Python-Code dargestellt, wie mit der lxml-Bibliothek eine XML-Datei erzeugt werden:

```
# Import lxml
from lxml import etree
# Erzeugen eines Elements
doc = etree.Element("root")
# Hinzufügen von einem Subelement
tree = etree.SubElement(doc, 'ergebnis')
# Hinzufügen von Subelementen
etree.SubElement(tree, 'feld', test='Test').text="Feld Eingabe"
# Erstellen eines Dokumenten Baums
doc = etree.ElementTree(doc)
# Speichern der XML Datei
doc.write("file.xml")
```

Mit diesem Code wird die „file.xml“ mit folgendem Inhalt geschrieben:

```
<root><ergebnis><feld test="Test">Feld Eingabe</feld></ergebnis></root>
```

Hierbei wird sichtbar, dass die XML-Elemente mit ihrem Anfangs- und Endtag, den Attributen und Werten automatisch erstellt werden. Zusätzlich gibt es auch noch die Möglichkeit, den Formatierungsstil über das Argument pretty\_print besser lesbar auszugeben. (RICHTER 2022)

Bei der lxml-Bibliothek ist es lediglich möglich, eine XML-Datei zu erzeugen, aber keine GML-Datei, da sich die Schemata der beiden verwandten Datei-Formate unterscheidet. Somit ist auch diese Bibliothek für den Export von XPlanGML-Dateien nicht verwendbar.

### 3.1.4 pygml-Bibliothek

Aber es gibt auch eine Bibliothek zur Erzeugung von GML-Daten, die pygml-Bibliothek. Bei dieser Bibliothek ist zu beachten, dass sie lediglich für das Lesen und Erzeugen von OGC GML-Geometrien genutzt werden kann. Ein Beispiel dafür, wie mit der Bibliothek GML-Daten aus GeoJSON erzeugt werden können, ist in der Abbildung 5 dargestellt. (SCHINDLER UND KRALIDIS 2022)

```

>>> from pygml.v32 import encode_v32
>>> from lxml import etree
>>> tree = encode_v32({'type': 'Point', 'coordinates': (1.0, 1.0)}, 'ID')
>>> print(etree.tostring(tree, pretty_print=True).decode())
<gml:Point xmlns:gml="http://www.opengis.net/gml/3.2" srsName="urn:ogc:def:crs:OGC::CRS84" gml:id="ID">
  <gml:pos>1.0 1.0</gml:pos>
</gml:Point>

```

Abbildung 5: Codebeispiel für die pygml-Bibliothek. (SCHINDLER UND KRALIDIS 2022)

Das Codebeispiel zeigt, dass mit der Bibliothek ein Ansatz für die Erzeugung von GML-Daten geschaffen wurde, der aber nur eingeschränkt für OGC GML-Geometrien eingesetzt werden kann. Für das Auslesen der GML-Daten wird hierbei auch auf die zuvor vorgestellte Bibliothek lxml zugegriffen. Dadurch, dass die Bibliothek nur GML-Geometrien erzeugen kann, ist auch diese Bibliothek für die Erzeugung von XPlanGML-Dateien nicht geeignet. Für die Erzeugung von Geometrien in den XPlanGML-Daten kann die Bibliothek ebenfalls so nicht genutzt werden, da diese Geometrien in den XPlanGML-Daten durch die Konformitätsbedingungen weiter eingeschränkt werden gegenüber dem OGC GML-Geometrien.

### 3.1.5 IO-Modul

Das IO-Modul ist das Kernwerkzeug in Python für das Arbeiten mit Datenströmen. Hierbei gibt es in dem Modul drei Typen von Datenströmen: Text, Binär und Roh (PYTHON SOFTWARE FOUNDATION 2022a). Mit einem Datenstrom (engl. data stream) wird die Ein- und Ausgabe von Daten in Dateien realisiert. Es wird bei den Streams zwischen dem Downstreams, den eingehenden Datenströmen, und den Upstreams, den ausgehenden Datenströmen, unterschieden (STEYER 2018a, S. 211 f).

Wichtig bei den Operationen der Datenströme ist, dass zu Anfang immer eine Datei geöffnet (open) und am Ende wieder geschlossen (close) wird. Mit der open()-Funktionen wird das Öffnen der Datei durchgeführt. Dabei muss der entsprechende Dateipfad angegeben werden sowie ein entsprechender Dateimodus. Es wird zwischen drei verschiedenen Modi unterschieden. Als erstes gibt es den Modus „r“, der für „read“ steht, dem Lesen der Datei. Der zweite Modus ist das „w“, das für „write“ steht. Mit diesem Modus wird eine Datei neu geschrieben. Der dritte Dateimodus ist das „a“ für „append“. Mit diesem Modus werden an bestehenden Inhalten neue Inhalte angehängt. Bei der close()-Funktion wird die Datei wieder geschlossen und die Bearbeitung ist damit dann beendet. Zum Schreiben der Datei gibt es in dem Modul die Funktion write(). Diese Funktion muss eine der zu Beginn genannten drei

Datentypen übergeben werden, der in die Datei geschrieben werden soll. Ein Beispiel für das Erzeugen einer Textdatei ist in Abbildung 6 dargestellt. Dabei wird ein String in eine Textdatei geschrieben. (STEYER 2018a, S. 212 ff)

```
fp = open("zitat1MP.txt", "w")
fp.write("Ich meine, was hast du schon zu verlieren? " +
        "Du weißt du kommst aus dem Nichts " +
        "und du gehst wieder ins Nichts zurück. " +
        "Was hast du also verloren? Nichts!")
fp.close()
```

Abbildung 6: Beispiel für das Schreiben eine Textdatei mit den Funktionen von dem io-Modul. (STEYER 2018a, S. 213)

Bei der Erzeugung von Dateien über Datenströme kann jede beliebige Datei-Endung verwendet werden. Wie in dem Beispiel der Abbildung 6 zu sehen ist, wurde die Endung .txt für eine Textdatei verwendet. Somit kann auch im Dateipfad die Endung .gml verwendet werden, um XPlanGML-Dateien erzeugen zu können. Beim Schreiben der Inhalte können ebenfalls Textdaten als String in XPlanGML-Form über die write-Funktion in die Datei gestreamt werden und somit erhält man am Ende eine GML-Datei mit den XPlanGML-Daten.

## 3.2 Methodik und Durchführung

In diesem Kapitel werden die Methodik und Durchführung für die Umsetzung der Exportschnittstelle für XPlanGML-Dateien in dem Plugin dargelegt. Dabei wird zuerst auf die Grundlage des vorhandenen Plugin XPlanung in QGIS eingegangen. Anschließend werden die Anforderungen aus den Konformitätsbedingungen für die Exportschnittstelle und abschließend die prototypische Umsetzung des Exports nach XPlanGML aufgezeigt. Dabei wird ein Schema aufgestellt, mit dem die Exportschnittstelle von anderen Programmierer\*innen zu Ende entwickelt werden kann.

### 3.2.1 Struktur des vorhandenen Plugins XPlanung

Wie schon im Kapitel 2.2.3 erwähnt, wurde von STRÖBL (2021a) das Plugin XPlanung mit dem dazugehörigen Plugin Data-Driven Input Mask in den SQL-Skripten für die PostgreSQL-Datenbank in QGIS entwickelt. In Abbildung 7 ist eine Übersicht zu den Skripten des XPlanung-Plugins mit den enthaltenden Klassen und Funktionalitäten. Hierbei wurde auch schon ein Skript für den Export der XPlanGML-Dateien in Abbildung 7 miteingefügt. Das gesamte Plugin

wurde mit der Programmiersprache Python umgesetzt. Die Erweiterung um das Tool des Exports nach XPlanGML wird in den vorhandenen Programmcode des Plugins mit eingebaut.

Hierbei sieht man an der obersten Stelle das Python-Skript `init.py` mit dem das ganze Plugin initialisiert wird. Dabei wird ein Objekt aus der Klasse `XPlan` erzeugt. Diese Klasse wird in der `XPlan.py` definiert und initialisiert die Tools des Plugins sowie die Anzeige der Menüpunkte in QGIS. Mit dem Skript `XPlanDialog.py` werden die GUIs zu den Tools realisiert. Hinter dem Skript `XPTool.py` sind einige Funktionen zu den Tools aus dem Plugin hinterlegt. Mit der Klasse `DbHandler` in dem Skript `HandleDb.py` wird die Datenbankverbindung realisiert. Die Funktionalitäten zu dem Import von XPlanGML-Dateien wird in der Klasse `XPImporter` in der `XPImport.py` ausgeführt. Da der Import sehr umfangreich ist, sind die Funktionalitäten für die Übersichtlichkeit in einem extra Skript mit einer eigenen Klasse ausgelagert worden und nicht in der Klasse `XPTools` implementiert.

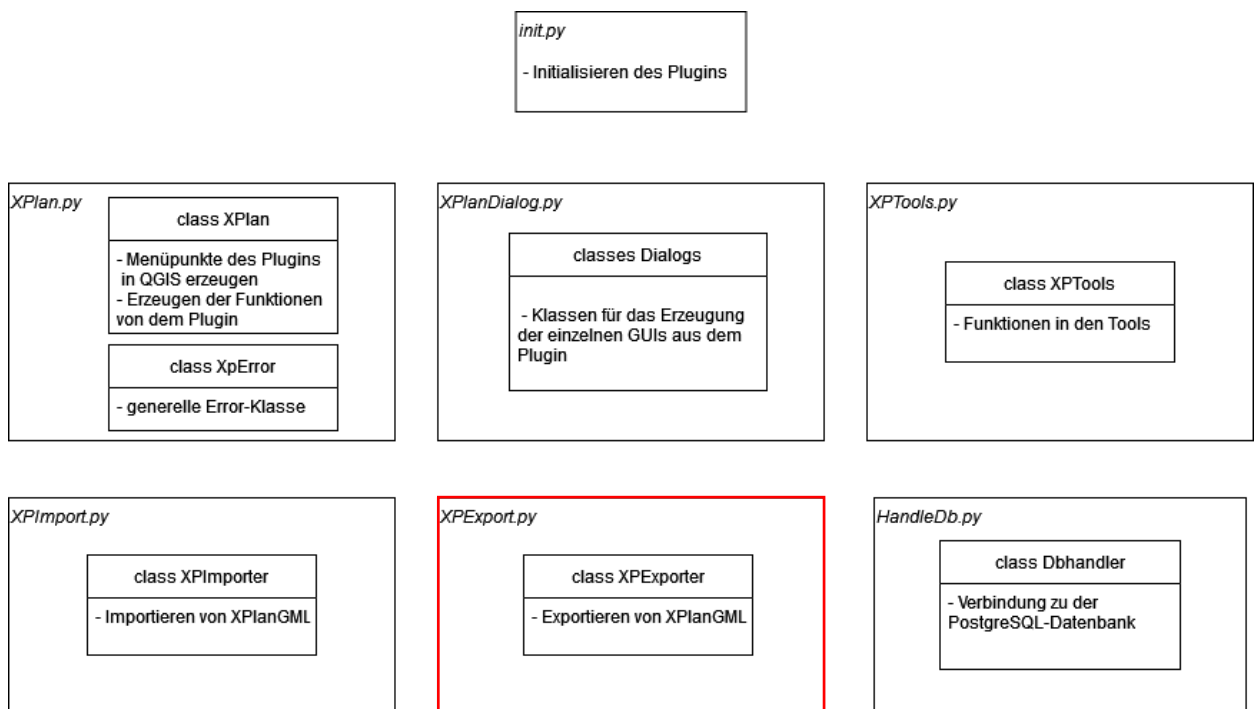


Abbildung 7: Übersicht der Python-Skripte mit den Klassen zu dem QGIS-Plugin XPlanung. (eigene)

Das dargestellte Skript `XPEXport.py` (roter Kasten) ist bis jetzt noch nicht in dem jetzigen Plugin von STRÖBL (2021a) entwickelt worden. Die Funktionalität des Exports wird in Rahmen dieser Arbeit in dem Skript `XPEXport.py` entwickelt. Das eigene Skript erhält die Funktionalität des Exports, da auch diese Funktion zu umfangreich für die Klasse `XPTools` ist.

Das XPlanung-Plugin wurde ähnlich der Model-View-Controller-Architektur (MVC) realisiert. Diese Architektur wird gerade bei Systemen angewandt, bei denen mehrere Views (Fenster) gleichzeitig aufgerufen werden und ist eines der meist verbreiteten objektorientierten Architekturmodelle. Beim MVC wird ein interaktives System in die drei Komponenten Model (Verarbeitung/Anwendungslogik), View (Ausgabe/Präsentation) und Controller (Benutzereingabe/Dialogsteuerung) unterteilt. Diese Bestandteile des interaktiven Systems werden voneinander getrennt, können aber einfach untereinander miteinander kommunizieren. Diese Kommunikation ist in Abbildung 8 mit den Abhängigkeiten dargestellt. Die Views und der Controller sind zusammen das User Interface (Benutzeroberfläche) und im Model wird die Logik des Systems umgesetzt. Das Model ist unabhängig von den Views und den Controller, da es lediglich die Eingaben aus dem User Interface verarbeitet. (GOLL 2014, S. 377 ff)

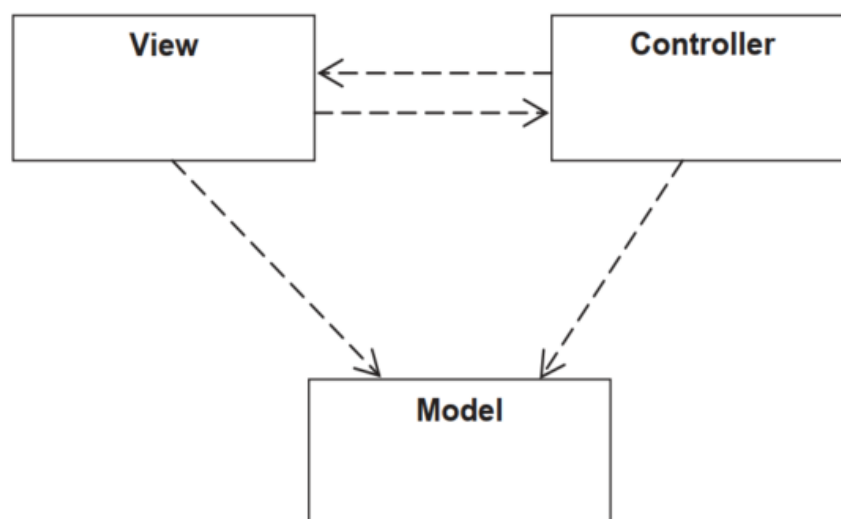


Abbildung 8: MVC-Architektur mit den Abhängigkeiten. (GOLL 2014, S. 380)

Somit musste auch die Exportschnittstelle in die MVC-Architektur des Plugins XPlanung eingebaut werden. Für den Controller des Exports wurde eine Funktionalität in die Klasse XPlan eingefügt. Für die verschiedenen Views des Exports wurden die jeweiligen GUIs in eigenen Klassen in dem Skript XPlanDialog erstellt. Da das Skript für den Export sehr umfangreich wird, wurde für die Anwendungslogik des Tools wie auch beim Import von XPlanGML eine eigene Klasse in einem eigenen Skript angelegt. Somit wurde das Model in einer extra Klasse entwickelt.

### 3.2.2 Anforderungen an den XPlanGML-Export

An eine XPlanGML-Datei werden verschiedene Anforderungen gestellt, damit diese Datei von dem XPlanValidator von KRAUSE (2022a) als valide eingestuft wird. Damit eine GML-Datei valide ist, muss zum einen die Struktur des XML-Schemas von der xsd-Dateien und zum anderen die semantischen sowie geometrischen Anforderungen an die XPlanGML-Dateien eingehalten werden. Diese Anforderungen an die XPlanGML-Datei werden in den Konformitätsbedingungen von BENNER (2020a) definiert; in dieser Arbeit beruhen alle auf XPlanung Version 5.2.1, da dies die aktuellste Version ist, auf die sich das Plugin in QGIS stützt (STRÖBL 2021a).

#### **Schema-Validität**

In dem Ordner des Releases für die XPlanung Version 5.2.1 auf der Webseite der XLeitstelle von KRAUSE (2022b) befindet sich für die Struktur des XML-Schemas ein Ordner mit den XSD-Schemadateien. XSD steht für XML Schema Definition und spezifiziert das XML/GML-Schema. Für die Beschreibung der Elemente und Attribute werden in der XSD-Datei element- und attribute-Elemente eingesetzt (BRINKHOFF 2013, S. 326 ff). Diese Schema-Dateien wurden mit der UmlToXmlTransformation-Software von der KIT aus den UML-Diagrammen zu den XPlanung-Objekten erzeugt.

Das Datenformat Geography Markup Language (GML) wurde von der Organisation Open Geospatial Consortium (OGC) als XML-basierte Repräsentation von Geodaten veröffentlicht (BRINKHOFF 2013, S. 338). Das OGC ist ein Zusammenschluss aus verschiedenen Akteuren aus dem Geoinformationsbereich, die mittels Standards die Interoperabilität zwischen Geoinformationssystemen herstellen wollen (BRINKHOFF 2013, S. 5). Ein solcher interoperabler Standard ist mit dem Dateiformat GML gegeben. Der Standard wurde von der OGC in verschiedenen Versionen veröffentlicht (BRINKHOFF 2013, S. 338). In dieser Arbeit ist nur die GML-Version 3.2.1 von Interesse, da das Datenaustauschformat XPlanGML auf diesem aufbaut (BENNER 2020a, S. 15). Diese OGC-Version von GML wurde auch als ISO-Norm 19136 verabschiedet. (BRINKHOFF 2013, S. 338)

In dem Ordner mit dem XML-Schemata befindet sich für das XPlanGML-Schema eine Oberdatei XPlanGML.xsd, aus der auf die anderen XSD-Dateien verwiesen wird. Als erstes wird auf die Datei XPlanGML\_Basisschema.xsd verwiesen (KRAUSE 2022b). In dieser Datei wird das



Basisschema mit den Abstrakten Oberklassen XP definiert, aus denen alle Klassen für die Schemata der Planwerke abgeleitet werden. Zusätzlich werden auch noch die allgemeinen Feature-Types, DataTypes sowie die Enumerations für die verschiedenen Planwerke definiert (KRAUSE 2020, S. 11). Nach dem Basisschema wird in der Oberdatei auf die Schema-Dateien der Fachschemata verwiesen, wie zum Beispiel für die Bebauungspläne die Datei XPlanGML\_BPlan.xsd (KRAUSE 2022b). In diesem Fachschema für die Bebauungspläne werden alle Bebauungsplan-Fachobjekte vorgehalten, die für die Modellierung aller Festsetzungen, Kennzeichnungen oder Vermerke in einem Bebauungsplan nach dem BauGB in den entsprechenden Klassen verwendet werden müssen (KRAUSE 2020, S. 24).

In Abbildung 9 sieht man einen Ausschnitt aus der XSD-Datei für die Bebauungsplan-Fachobjekte. In dem Ausschnitt der Abbildung ist die Klasse BP\_Bereich mit den dazugehörigen Attributen abgebildet. Anhand dieser Schema-Daten können für die XPlanGML-Daten die entsprechenden Objekte mit den richtigen Datentypen zu den Attributen erzeugt werden. Um nachzuvollziehen, aus welcher Oberklasse sich die dargestellte Klasse ableitet, wird in der vierten Zeile unter dem Tag „element“ im Attribut „substitutionGroup“ die Oberklasse angegeben. Diese Klasse muss in den GML-Daten für das Objekt vorher realisiert werden.

Mit dem ersten Element in der Schema-Datei wird der Objekt-Name angegeben. Mit den danach folgenden Elementen werden die Attribute zu dem Objekt definiert. Der Name des Elements ist der Name des Tags in den GML-Daten und mit der Angabe „type“ wird der Datentyp vorgeschrieben. Anhand des Datentyps können auch noch weitere Element-Attribute definiert werden, wie z. B. die Angabe von Maßeinheit für den Attributtyp „gml:LengthType“, der in dem Beispiel der Abbildung allerdings nicht vorhanden ist. Des Weiteren ist in der Abbildung unter dem Element „gehörtZuPlan“ der Datentyp „gml:ReferenzTyp“ zu sehen, wobei mit einem Verweis über die GML-ID auf das dazugehörige Plangebiet eine Referenz erzeugt wird.



Dafür ist bei der Programmierung zu beachten, dass nicht die normale GML 3-Geometrien verwendet werden, da diese durch die entsprechende XSD-Datei und die Konformitätsbedingungen angepasst werden. So können sowohl die aus den Datenbanken abgefragten GML-Geometrien über die Funktion „ST\_AsGML“ als auch die pygml-Bibliothek mit der GML 3-Geometrien erzeugt werden können, nicht genutzt werden (POSTGIS DEVELOPMENT GROUP 2022b; SCHINDLER UND KRALIDIS 2022).

Für die klassenspezifischen Konformitätsbedingungen werden einerseits für die abstrakten Oberklassen und andererseits für die abgeleiteten Klassen der Planwerke Regeln und Einschränkungen definiert. Dabei handelt es sich zum Beispiel um Relationen oder Rückwärts-Referenzen zu Objekten. Es werden aber auch Einschränkungen bei den Attributen angegeben, wobei z. B. eines der Attribute belegt sein muss, aber nicht beide gleichzeitig belegt sein dürfen. (BENNER 2020a, S. 17 ff)

Die klassenspezifischen Konformitätsbedingungen werden bei der Entwicklung der Exportschnittstelle nicht berücksichtigt und müssen somit von den Benutzer\*innen des Plugins selber bei der Erzeugung der Sachdaten zu den XPlanung-Objekten berücksichtigt werden. Ein weiteres fehlendes Tool in dieser Hinsicht ist die Prüfung der Daten in einem Plan auf die semantischen Bedingungen, die mit den Konformitätsregeln aufgestellt werden. Diese Funktion ist gerade im Rahmen des Exports wichtig, um die Regeln für XPlanGML-Dateien zu prüfen.

Somit wird deutlich, dass an den Entwickler\*innen der Exportschnittstelle nach XPlanGML-Dateien einige Anforderungen in Bezug auf die syntaktischen, semantischen und geometrischen Inhalte gestellt werden. Gerade die syntaktischen und geometrischen Inhalte müssen zwingend in der Entwicklung des Exports berücksichtigt werden. Dabei können die Regeln der semantischen Inhalte auch bei der Eingabe von den Benutzer\*innen berücksichtigt werden.

### 3.2.3 Prototypische Entwicklung des Tools zum XPlanGML-Export

In diesem Kapitel wird die prototypische Entwicklung des Tools der Exportschnittstelle nach XPlan-GML vorgestellt. Im Rahmen dieser Arbeit ist es nicht möglich, das gesamte Tool zu entwickeln, da die Objekte zu den ganzen Planwerken sehr umfangreich sind. Deshalb wird lediglich ein Teil des Export-Tools prototypisch entwickelt. Dies inkludiert die Erarbeitung

einer Struktur für den Export sowie ein Schema, mit dem die Exportschnittstelle weiterentwickelt werden kann. Hierfür ist in Abbildung 10 ein UML-Aktivitätsdiagramm zu sehen, anhand dessen der Ablauf des Export-Tools erläutert wird.

Die prototypische Entwicklung des Exports wurde im Rahmen dieser Arbeit nur für die XPlanung-Version 5.2, für Bebauungspläne (BP) und nicht für alle Objekte umgesetzt. Für die anderen Optionen bei der XPlanung-Version und Planart gelangt das Tool in einen Endzustand, in dem Benutzer\*innen mitgeteilt wird, dass der Export hierfür noch nicht umgesetzt wurde. Der Export für Bebauungspläne wurde nur für folgenden XPlanung-Objekte umgesetzt: BP\_Plan, BP\_Bereich, BP\_BaugebietsTeilFlaeche und BP\_StrassenVerkehrsFlaechen. BP\_Plan und BP\_Bereich müssen immer in XPlanung für einen Bebauungsplan angegeben werden, da diese die Grundlage für die Pläne bilden. Für die beiden Klassen BP\_BaugebietsTeilFlaeche und BP\_StrassenVerkehrsFlaechen wurde zur näheren Betrachtung ausgewählt, da dies klassische Objekte für Baugebiete in Bebauungspläne sind und die Umsetzung anhand dessen gut erläutert werden kann. Die XPlanung-Objekte wurden mit Hilfe der vorliegenden XSD-Schemadateien „XPlanGML\_Basisschema.xsd“ und „XPlanGML\_BPlan.xsd“ erzeugt.

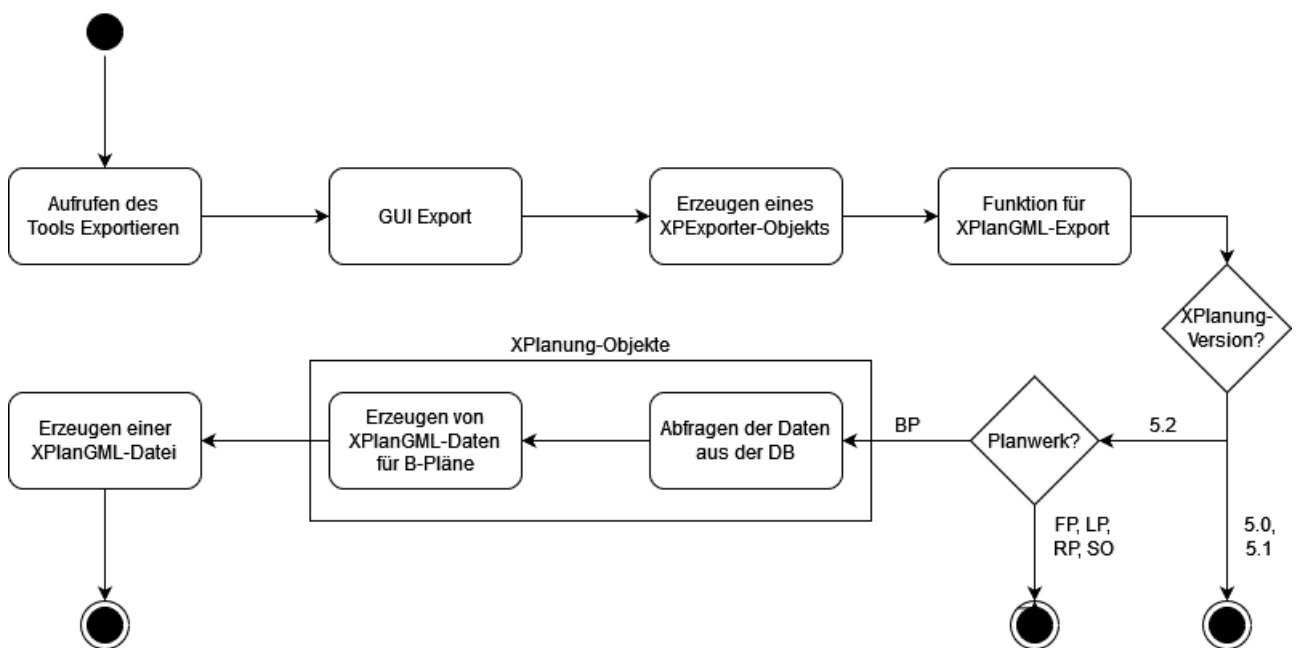


Abbildung 10: UML-Aktivitätsdiagramm des Tools Exportieren in QGIS. (eigene)

Wie in der Abbildung 10 erkennbar ist, wird nach der Auswahl des XPlanung-Tools „Exportieren“ in QGIS die GUI für den Export geöffnet. In dieser GUI werden alle Angaben zum Export der XPlanGML-Datei gemacht. Mit diesen Angaben wird das Objekt XPEXporter erzeugt,

indem die Funktion zum Export von XPlanGML-Dateien aufgerufen wird. Anschließend werden die Entscheidungsknoten zu der Version und dem Planwerk durchlaufen. Nach den Entscheidungen werden für die XPlanung-Objekte die Daten aus der Datenbank abgefragt und in die XPlanGML-Daten eingebaut. Im abschließenden Schritt werden die GML-Daten in eine GML-Datei geschrieben. Diese Schritte des UML-Diagramms werden mit der Umsetzung in den nachfolgenden Abschnitten näher beschrieben.

## **GUI Export**

In Abbildung 11 und Abbildung 12 sind die graphischen Benutzerflächen (GUI) für den Export dargestellt. Diese GUIs wurden mit der Software Qt Creator erstellt. Für die Gestaltung der Oberflächen dienten die vorhandenen Tools aus dem Plugin XPlanung als Orientierung. In den GUIs sind keine Einschränkungen eingebaut, obwohl nicht alle XPlanung-Versionen und Planwerke realisiert wurden. Die Realisierung der GUIs wurde so umgesetzt, wie diese in einem vollständigen Export aussehen würden. Wenn die nicht realisierten Auswahlmöglichkeiten in den GUIs ausgewählt werden, landet man in einem Endzustand, wie in dem UML-Diagramm zu sehen, und erhält eine Warnung, dass die Funktionalitäten noch nicht umgesetzt sind.

Abbildung 11 zeigt das Hauptfenster für den Export, das zu Beginn geöffnet wird. In diesem Fenster können die Benutzer\*innen alle wichtigen Angaben für den Export eingeben. Im oberen Bereich ist die Auswahl der Plangebiete realisiert. In dem Kasten wird eine Angabe zu dem Planwerk und dem Namen des zu exportierenden Plangebiets angegeben. Wenn noch kein Plangebiet ausgewählt wurde oder das Plangebiet geändert werden soll, muss der Button „ändern“ auf der rechten Seite des Kastens verwendet werden. Über diesen Button gelangt man in die GUI aus Abbildung 12, in dem die Plangebiete ausgewählt werden können.

In der Mitte der GUI für den Export befindet sich ein Dropdown-Fenster, in dem die XPlanung-Version angegeben werden kann. Sie ist standardmäßig auf die höchste und damit aktuellste Version eingestellt. Dies ist in dem Fall des Plugins die XPlanung-Version 5.2.

Im unteren Teil der GUI ist ein Textfeld für die Eingabe des Speicherpfads der XPlanGML-Datei. Mit dem Button „...“ neben dem Textfeld wird über die Klasse QFileDialog ein Dialogfenster geöffnet, mit dem der Speicherort ausgewählt und der Dateiname angegeben werden kann. Bei der Angabe von dem Dateinamen ist es möglich, ein Dateiformat vorzugeben, damit die

Dateipfade immer auf diesem Dateiformat enden müssen. Somit wurde dieses Dateiformat auf GML festgelegt, sodass bei der Eingabe immer eine XPlanGML-Datei erzeugt wird. Dies ist in Abbildung 11 mit einem Beispieldateipfad dargestellt. (THE QT COMPANY 2022)

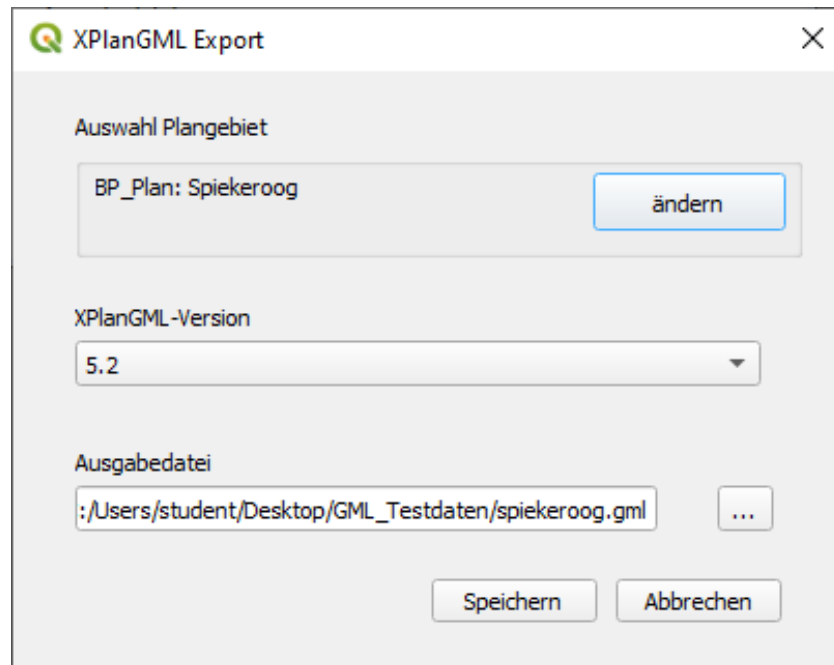


Abbildung 11: GUI für den Export von XPlanGML-Dateien aus QGIS. (eigene)

Wie schon zuvor erwähnt, ist in Abbildung 12 die GUI für die Gebietsauswahl für den Export dargestellt. Die GUI für die Plangebietsauswahl lehnt sich an der GUI für die Bereichsauswahl der Funktionalität „Bereich laden“ an, da sie eine ähnliche Funktionsweise aufweisen. Hier kann im unteren Abschnitt der GUI eine Auswahlbeschränkung durchgeführt werden, indem ein bestimmtes Planwerk über Radiobuttons ausgewählt wird. Zu diesem Planwerk werden dann alle vorhandenen Plangebiete in dem oberen Bereich der GUI angegeben und das gewünschte Plangebiet kann hier ausgewählt werden. Die Auswahl der Plangebiete kann über einen Doppelklick oder der Bestätigung der Auswahl über den OK-Button durchgeführt werden. Hierbei liegt der Fokus bewusst auf der Einschränkung der Auswahl auf die Plangebiet, da für die prototypische Umsetzung in dieser Arbeit auch die großen Einheiten der Plangebiete ausreichen. Die Umsetzung für eine Auswahl für die Bereich-Ebene ist jedoch relativ einfach umsetzbar und kann auf dieser Grundlage aufbauen. Mit der Bestätigung der Auswahl wird dies in die GUI des Exports übertragen wie zuvor beschrieben.

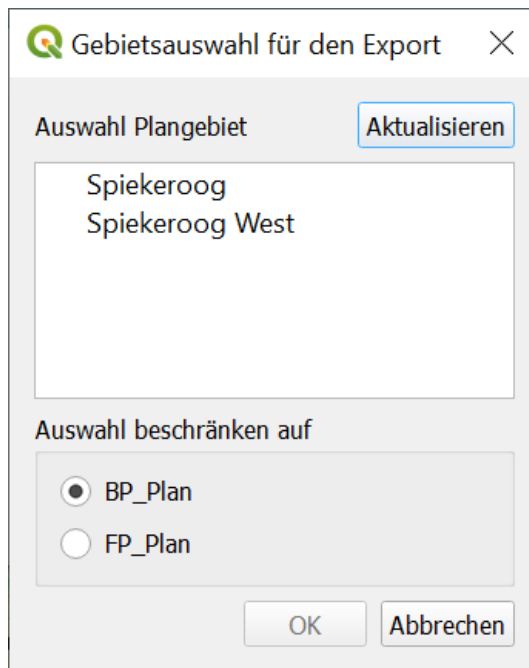


Abbildung 12: GUI für die Auswahl des Plangebiet für den Export. (eigene)

Nur wenn alle Auswahlmöglichkeiten von den Benutzer\*innen getätigt wurden, können diese mit dem Button „Speichern“ in der GUI „XPlanGML Export“ bestätigt werden. Fehlen noch Angaben in der GUI, so bleibt der „Speichern“-Button ausgegraut. Diese Angaben aus der Benutzeroberfläche werden an das XPEXporter-Objekt übergeben. Mit diesem Objekt wird das Tool zum Export von XPlanGML-Dateien realisiert. Dabei gelangt man danach in dem UML-Aktivitätsdiagramm an zwei Entscheidungsknoten. Bei dem ersten Entscheidungsknoten wird entschieden, mit welcher XPlanung Version die Datei exportiert werden soll. Wenn eine andere Version als 5.2 ausgewählt wird, erscheint nach der Bestätigung dieser Eingabe der Hinweis, dass diese Version noch nicht in dem Tool umgesetzt wurde und der Export wird nicht durchgeführt werden kann. Bei dem nachfolgenden zweiten Entscheidungsknoten wird das Planwerk des zu exportierenden Plans abgefragt. Auch hier ist die Auswahl eingeschränkt, da der prototypische Export nur für die Bebauungspläne umgesetzt wurde. Bei den anderen Planwerken bietet sich ebenfalls die Meldung dar, dass der Export für diese Planwerke noch nicht umgesetzt wurde.

### Erzeugen der GML-Daten

Anschließend kann mit dem Erzeugen der GML-Daten gestartet werden. Dies wird mit der Funktion „exportBP\_5\_2“ für die Bebauungspläne der XPlanung-Version 5.2 durchgeführt. Aus dieser Funktion werden in diesem Abschnitt einige wichtige Programmcodebeispiele

erläutert, mit denen ein Schema abgeleitet wird und anhand dessen auch die noch fehlenden Objekte erzeugt werden können. In der String-Variablen „gml\_BP“ der Funktion werden die GML-Daten reingeschrieben.

Nach diesem Abschnitt ist der Code zu sehen, in dem der Header der GML-Daten sowie der Anfangs- und End-Tag definiert werden. Dieser wurde mit Hilfe der XSD-Datei „XPlanung-Operationen.xsd“ erzeugt. Im Anfangs-Tag „xplan:XPlanAuszug“ ist zu sehen, dass einige Angaben zu dem WFS-, GML-, XPlanung-Version und Ähnlichem gemacht werden. Des Weiteren muss bei den Eigenschaften zu dem Element XPlanAuszug auch noch eine GML-ID vergeben werden. Da diese einzigartig für den XPlanAuszug sein muss, wird dies mit der Bibliothek uuid und der Funktion „uuid4()“ generiert. Mit dieser Funktion wird eine zufällige und sichere ID erzeugt (PYTHON SOFTWARE FOUNDATION 2022c). Mit den Funktionen von der Bibliothek werden in den Plugin für neue XPlanGML-Objekte die GML-IDs vergeben.

Zwischen dem Anfangs- und End-Tag werden an der Stelle des Platzhalters „{...}“ die weiteren XPlanGML-Daten für die Objekte aus Funktionen definiert. Dieser Platzhalter wird im Laufe der Arbeit immer wieder verwendet, um den dargestellten Programmcode auf ein Minimum zu kürzen. Der vollständige Programmcode ist im Anhang hinterlegt.

```
# Header Daten
gml_BP = '<?xml version="1.0" encoding="utf-8"?>\n'+\
# Anfangstag des Elements XPlanAuszug
  '<xplan:XPlanAuszug xmlns:wfs="http://www.opengis.net/wfs"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink" gml:id= "Gml_'+str(uuid.uuid4())+' "
xsi:schemaLocation="http://www.xplanung.de/xplangml/'+ xPlan_xsd1 +'
http://www.xplanungwiki.de/upload/XPlanGML/'+ self.params["xsdNr"]
+'/Schema/XPlanung-Operationen.xsd"
xmlns:xplan="http://www.xplanung.de/xplangml/'+ xPlan_xsd1 +' ">\n'
{...}
# Ende der GML, Schließen des Elements XPlanAuszug
gml_BP += "</xplan:XPlanAuszug>"
```

Als erstes wird für das Objekt „xplan:XPlanAuszug“ mit dem GML-Tag „gml:boundedBy“ das minimal umgebende Rechteck (MUR) definiert. Da auch für andere Objekte noch ein minimal umgebendes Rechteck benötigt werden, wurde dies mit der nachfolgenden Funktion extra umgesetzt:



```

# Kleinstes Recht (boundedBy) um Objekt
def exp_boundedBy(self, tab, koor_loCorner, koor_upCorner):
    boundBy = tab * str("\t") + "<gml:boundedBy>\n"
    boundBy += (tab+1) * str("\t") + '<gml:Envelope srsName="' + self.epsg + '">\n'
    boundBy += (tab+2) * str("\t") +
"<gml:lowerCorner>" + koor_loCorner + "</gml:lowerCorner>\n"
    boundBy += (tab+2) * str("\t") +
"<gml:upperCorner>" + koor_upCorner + "</gml:upperCorner>\n"
    boundBy += (tab+1) * str("\t") + "</gml:Envelope>\n"
    boundBy += tab * str("\t") + "</gml:boundedBy>\n"
    return boundBy

```

Bei der Funktion sind die übergebenen Variablen in der Klammer zu sehen. Mit der Variablen „tab“ wird die Anzahl der Tab-Einrückungen angegeben. Theoretisch kann eine GML-Datei auch ohne Tabs („\t“) und Zeilenumbrüchen („\n“) erzeugt werden, aber über diese Einrückungen und Zeilenumbrüche wird die Lesbarkeit der GML-Daten verbessert, da die Tags je nach ihrer Hierarchieebenen eingerückt werden und jedes Attribut erhält eine eigene Zeile. Die beiden weiteren Variablen „koor\_loCorner“ und „koor\_upCorner“ sind die Angaben der Koordinaten für die unteren und oberen Ecken des MURs. Diese wurden zuvor mit einer Datenbankabfrage abgerufen. Diese Angaben werden anschließend in das entsprechende GML-Format als String übergeben. Am Ende wird der String „boundBy“ mit den GML-Daten aus der Funktion wieder ausgegeben.

### Erzeugen der GML-Daten zu den XPlanung-Objekten

Die weiteren Objekte im XPlanAusschnitt wurden mit Hilfe der XML-Schemas „XPlanGML\_Basisschema.xsd“ für die abstrakten Oberklassen und „XPlanGML\_BPlan.xsd“ für das XML-Schema des Planwerks der Bebauungspläne erzeugt. In dieser Arbeit werden lediglich die Klassen BP\_Plan, BP\_Bereich, BP\_BaugebietsTeilFlaeche und BP\_StrassenVerkehrsFlaechen für den Export realisiert, da die Entwicklung von weiteren Klassen den zeitlichen Umfang dieser Arbeit überschreiten würden.

In diesem Abschnitt wird beispielhaft die Erzeugung des ersten Objektes BP\_Plan für die Bebauungspläne dargestellt, da diese ähnlich auf die anderen Objekte übertragen werden kann. Der folgende Code nach diesem Abschnitt zeigt, dass für das Objekt BP\_Plan die Funktion exp\_plangebietGML aufgerufen wird und als Ergebnis der GML-String aus der Funktion an die String-Variable gml\_BP angehängt wird. Alle Funktionen, in denen XPlanGML-Daten erzeugt werden, haben in dem Funktionsnamen am Anfang das Kürzel

„exp\_“. Bei diesem Objekt muss keine Abfrage nach der Anzahl der Objekte durchgeführt werden, da nach der Definition in der GUI immer nur ein Plangebiet ausgewählt werden kann. Bei den anderen Objekten muss immer überprüft werden, wie viele Objekte in dem gesamten Plangebiet vorliegen. An die Funktion werden die Koordinaten des MURs übergeben, da diese bereits zuvor mit einer Datenbankabfrage abgerufen wurden und für das Gebiet übereinstimmen. Alle anderen Datenbankabfragen werden in der Funktion durchgeführt.

```
# BP_Plan
gml_BP += self.exp_plangebietGML(koor_loCorner, koor_upCorner)
```

In der Funktion `exp_plangebietGML` werden zuerst mittels Funktionen die Attribute für das Objekt `BP_Plan` von der Datenbank abgefragt. In dem nachfolgenden Programmcode ist zu sehen, dass die Daten aus der Datenbank für die Attribute von der abstrakten Oberklasse `XP_Plan` und der Klasse `BP_Plan` mit verschiedenen Funktionen abgefragt werden. Da bei einigen Attributen eine n:m-Beziehung vorliegt und somit die Daten in extra Tabellen abgespeichert sind, müssen diese mit separaten Funktionen abgerufen werden. Alle Funktionsnamen für Datenbankabfragen beginnen in dem Programmcode mit dem Kürzel „abf\_“ und erhalten danach zusätzlich den Namen der abgefragten Tabelle. Somit kann nachvollzogen werden, ob Abfragen an Tabellen bereits in einer Funktion umgesetzt wurden und es entstehen keine Redundanzen bei den Abfrage-Funktionen.

Bei der letzten Abfrage-Funktion in dem dargelegten Code werden keine Werte an eine Variable übergeben, da die abgefragten Attribute für weitere XPlanung-Objekte benötigt werden. Aus diesem Grund werden in dieser Funktion globale Variablen erzeugt, sodass die Attribute auch in anderen Funktionen verwendet werden können.

```
# Abfragen für die Attribute der Datenbank
# Abfrage XP_Plan
xp_plan = self.abf_xpPlan()
aendListe = self.abf_aendertPlan()
wGeaeVon = self.abf_wgvPlan()
rGeltungB = self.abf_raeumlGeltPlan()
verfMerk = self.abf_verfMerkPlan()
tex = self.abf_texte()
begAbschnitt = self.abf_begAbsch()

# Abfrage BP_Plan
bp_Plan = self.abf_bpPlan()
gemeinde = self.abf_gemeinde()
```

```

planaufGem = self.abf_planaufGemeinde()
planArt = self.abf_planArt()
self.abf_bereichGMLID() # Hier werden globale Variablen definiert

```

In dem nächsten Codeausschnitt ist eine Funktion zur Datenbank-Abfrage der Attribute des Objekts XP\_Plan zu sehen. Die Abfrage-Funktionen sind ähnlich aufgebaut. Dabei wird in einem String als erstes die SQL-Abfrage definiert. Anschließend wird mit dem globalen Objekt db aus der Klasse DbHandler eine Verbindung mit der Datenbank aufgebaut. Wenn die Verbindung zur Datenbank aufgebaut ist, kann über die Funktion „prepare()“ die SQL-Abfrage übergeben und vorbereitet werden. Darauffolgend wird mit der Funktion „exec\_()“ die Datenbankabfrage ausgeführt. Bevor die Attribute aus der Datenbank abgerufen werden können, muss noch überprüft werden, ob die Abfrage aktiv ist mit der Funktion „isActive()“. Zusätzlich wird noch dafür Sorge getragen, dass die Länge des Inhalts nicht null ist und abschließend wird dann mit einer while-Schleife über die Daten iteriert (QT COMPANY 2020). Bei der Iteration über die Daten muss immer die Kardinalität der Attribute in der Datenbank berücksichtigt werden. Die Kardinalitäten zu den Attributen sind in dem Objektartenkatalog von KRAUSE (2020) hinterlegt. Bei dieser Abfrage ist die Kardinalität aller Attribut [0..1] oder [1], somit können die Attribute aus der Datenbank direkt an ein Variable übergeben werden, wie es hier in dem Code-Beispiel zu sehen ist. Wenn die Attribute die Kardinalität [0..\*] haben, müssen die Attribute in Listen abgespeichert werden, da hier unendliche viele Attribute vorliegen können.

```

# Abfrage der XP_Plan Attribute
def abf_xpPlan(self):
    xp_Plan_sql = "SELECT * FROM \"XP_Basisobjekte\".\"XP_Plan\" WHERE gid =
"+ self.gid +""
    xp_Plan_query = QSqlQuery(self.db)
    xp_Plan_query.prepare(xp_Plan_sql)
    xp_Plan_query.exec_()
    if xp_Plan_query.isActive():
        if xp_Plan_query.size() != 0:
            while xp_Plan_query.next():
                num = xp_Plan_query.value(2)
                intID = xp_Plan_query.value(3)
                beschr = xp_Plan_query.value(4)
                komm = xp_Plan_query.value(5)
                {...}
    return [num, intID, beschr, komm, {...}]

```

Nachdem alle Attribute mit den Abfragen aus der Datenbank abgerufen wurden, können die GML-Daten für das Objekt BP\_Plan erzeugt werden. Hierbei gibt es für die XPlanung-Objekte zwei Tags zu Beginn eines Objekts: zum einen das „gml:featureMember“, mit dem ein XPlanung-Objekt eingeleitet wird und zum anderen der Tag, der das XPlanung-Objekt definiert, hier „xplan:BP\_Plan“. Dabei wird bei dem GML-Objekt „xplan:BP\_Plan“ das Attribut „gml:id“ für die eindeutige Identifikation des Plangebiets angegeben. Zusätzlich wird auch bei den BP\_Plan-Objekten ein MUR über die Funktion „exp\_boundBy()“ erzeugt. Nur bei den Plan-Objekten muss immer ein MUR angegeben werden. Damit wurden alle allgemeinen Informationen zu den BP\_Plan-Objekt angegeben. Anstelle des Platzhalters „{...}“ werden für diese Objekt alle vorhandenen Attribute aufgelistet, die aus der Datenbank abgefragt wurden. Wenn kein Attribut in der Datenbank vorhanden ist, wird dieses Element auch nicht in die GML-Daten geschrieben. Dies wird beispielhaft in den nachfolgenden Code-Beispielen beschrieben, wobei zu Beginn die Attribute der Oberklasse und anschließend die Attribute der eigentlich Klasse definiert werden.

```
# Anfang-Tag BP_Plan
self.gmlID_plan = xp_plan[11]
planGebiet = '\t<gml:featureMember>\n'
planGebiet += '\t\t<xplan:BP_Plan gml:id="GML_'+ self.gmlID_plan +'>\n'
tab=3
planGebiet += self.exp_boundBy(tab, koor_loCorner, koor_upCorner)
{...}
# Ende des Plangebiets
planGebiet += '\t\t</xplan:BP_Plan>\n'
planGebiet += '\t</gml:featureMember>\n'
```

Bei den Attributen für das Objekt werden die Attribute aus verschiedenen Klassen abgeleitet. Als erstes müssen die Attribute der abstrakten Oberklasse angegeben werden, abschließend kommen die weiterabgeleiteten Klassen und die eigenen Attribute der Klasse. Für die Oberklassen wird es so gehandhabt, dass diese in eigenen Funktionen realisiert werden. Bei vielen abgeleiteten Oberklassen ist es der Fall, dass diese auch in anderen Objekten enthalten sind. Somit kann auf diese Funktionen zu den Klassen auch zugegriffen werden und es entstehen auch hier keine Redundanzen im Code.

In dem nachfolgenden Code wird für das Objekt BP\_Plan die Funktion der abstrakten Oberklasse XP\_Plan aufgerufen. Diese Funktion heißt „exp\_XP\_plan\_gml()“ und ihr werden die Variablen mit den abgefragten Attributen aus der Datenbank übergeben. Mit der Ausgabe

dieser Funktion wird ein String mit den dazugehörigen GML-Daten zu der Klasse XP\_Plan übergeben und den GML-Daten des Objekts BP\_Plan angehängt.

```
# XP_Plan
planGebiet += self.exp_XP_plan_gml(xp_plan, aendListe, wGeaeVon, rGeltungB,
verfMerk, tex, begAbschnitt)
```

Wie diese Funktionen den GML-String für die Attribute der Klasse zusammenstellen, ist in den anschließenden Code beispielhaft für das XP\_Plan-Klasse zu sehen. Die Attribute werden an die String-Variable in GML-Form übergeben. Für das erste Attribut „xplan:name“ muss keine Abfrage erzeugt werden, ob dieses Attribut vorhanden ist, da dieses Attribut immer vorhanden sein muss. Bei den weiteren Attributen des Objekts wird zuvor mit der if-Abfrage kontrolliert, ob ein Wert für das Attribut aus der Datenbank vorhanden ist. Wenn dies der Fall ist, wird das Attribut mit dem Wert als GML-Element angelegt. Diese Abfrage und Übergabe der Attribute an die GML-Daten werden mit allen Objekt-Attributen durchgeführt und abschließend werde die GML-Daten zum Objekt XP\_Plan ausgegeben.

```
# XP_Plan
def exp_XP_plan_gml(self, xp_plan, aendListe, wGeaeVon, rGeltungB, verfMerk,
tex, begAbschnitt):
    xp_plan_gml = ""
    xp_plan_gml += '\t\t\t<xplan:name>'+ self.params["plangebiet"] +
'</xplan:name>\n'
    if str(xp_plan[0]) != "NULL":
        xp_plan_gml += '\t\t\t<xplan:nummer>'+xp_plan[0]+'</xplan:nummer>\n'
    if str(xp_plan[1]) != "NULL":
        xp_plan_gml += '\t\t\t<xplan:internalId>'+ xp_plan[1] +
'</xplan:internalId>\n'
    {...}
    return xp_plan_gml
```

Für die eigenen Attribute der Klasse BP\_Plan wurden die Attribute der Klasse nach der Angabe der Attribute aus den Oberklassen wie im nachfolgenden Programmcode übergeben. Dabei werden die Attribute in der Funktion direkt realisiert. Wie man für das erste Attribut „xplan:gemeinde“ sieht, wird ein eigenes Objekt „XP\_Gemeinde“ über die Funktion „exp\_XPgemeinde\_gml“ angelegt. Somit wird deutlich, dass Attribute der Objekte nicht nur Werte, sondern auch XPlanung-Objekte enthalten können. Wenn Objekte an ein Attribut übergeben werden, ist dies entsprechend in den XSD definiert.

```

# BP_Plan
# BP_Gemeinde
if len(gemeinde)>0:
    for gem in gemeinde:
        planGebiet += '\t\t\t<xplan:gemeinde>\n'
        planGebiet += self.exp_XPgemeinde_gml(tab, gem)
        planGebiet += '\t\t\t</xplan:gemeinde>\n'
# BP_planaufstellendeGemeinde
if len(planaufGem)>0:
    for plangem in planaufGem:
        planGebiet += '\t\t\t<xplan:planaufstellendeGemeinde>\n'
        planGebiet += self.exp_XPgemeinde_gml(tab, plangem)
        planGebiet += '\t\t\t</xplan:planaufstellendeGemeinde>\n'
{...}
# Planart
if planArt != NULL:
    planGebiet += '\t\t\t<xplan:planArt>'+ str(planArt) + '</xplan:planArt>\n'
# sonstPlanArt
if str(bp_Plan[1])!="NULL":
    soPlanArt = self.abf_sonstPlanArt(bp_Plan[1])
    planGebiet += '\t\t\t<xplan:sonstPlanArt>'+ soPlanArt
+ '</xplan:sonstPlanArt>\n'
{...}

```

Nach dem bis jetzt vorgestellten Schema können fast alle Attribute für die GML-Daten in einer XPlanGML-Datei erzeugt werden. Nur bei den GML-Geometrien erfordert es ein weiteres Schema für die Erzeugung.

Für die Erzeugung der Geometrie in GML-Form müssen einige Einschränkungen beachtet werden, die in dem Kapitel 3.2.2 vorgestellt wurden. Somit ist es nicht möglich die Geometrien aus der PostGIS-Datenbank direkt über den Befehl „ST\_AsGML“ mit einer SQL-Abfrage als GML-Geometrie abzurufen (BRINKHOFF 2013, S. 349). Bei diesem Befehl ist es nicht möglich die Einschränkungen sowie Angabe der Attribute und Eigenschaften der GML-Geometrie umzusetzen.

Aus diesem Grund musste auch die Erzeugung der GML-Geometrien anhand des vorgegebenen XML-Schemas mit der Datei „gmlProfilexplan.xsd“ in einer eigenen Funktion realisiert werden. Dies wurde nur beispielhaft für den GML-Geometriotyp Polygone in dieser Arbeit umgesetzt. Dabei müssen die Geometrien mit ihren Attributen in einem auslesbaren Format aus der Datenbank abgefragt werden. Für die Aufbereitung der Daten soll das GeoJSON-Format verwendet werden. Dieses Format für die Geometrien kann mit der

Funktion „ST\_AsGeoJSON“ aus der Datenbank abgefragt werden (POSTGIS DEVELOPMENT GROUP 2022a). Um aus der Abfrage in Python eine JSON (JavaScript Objekt Notation) zu erzeugen, wird mit der Funktion „json.load()“ aus dem abgefragten String ein GeoJSON für die Geometrie erzeugt (PYTHON SOFTWARE FOUNDATION 2022b). Anschließend wird zum Auslesen der Daten die GeoJSON der Geometrie an die Funktion „geom\_Best()“ übergeben und aus diesen JSON-Daten die relevanten Daten für die GML-Daten extrahiert. Dabei werden die Daten zum Geometriety, die Daten zur Dimension der Koordinaten, die Koordinaten des Außen- und Innenbereichs des Polygons sowie die Anzahl dieser Koordinaten ausgegeben. Bei der Ausgabe der Koordinaten des Polygons gibt es die Besonderheit, dass die Reihenfolge der Koordinaten einen korrekten Umlaufsinn haben muss. In den Konformitätsbedingungen ist definiert, dass die Koordinaten von Außenkonturen Gegen-Uhrzeigersinn und für Inselflächen im Uhrzeigersinn sein müssen (BENNER 2020a, S. 16 f). Bei dieser beispielhaften Umsetzung der GML-Geometrien wurde lediglich die Koordinatenreihenfolge Gegen-Uhrzeigersinn für Außenkonturen umgesetzt. Somit ist auch im Programmcode zu sehen, dass die Koordinatenreihenfolge bei der Ausgabe umgedreht wird, da die Speicherung der Koordinaten in der Datenbank immer mit dem Uhrzeigersinn geschieht.

Nachdem die Bestandteile selektiert wurden, wird in der Funktion zur Erzeugung der GML-Geometrie abgefragt, welcher Geometriety vorliegt. Dies muss in diesem Fall immer ein Polygon sein, da die Funktion bis jetzt nur für Polygone umgesetzt wurde. Anschließend werden die Bestandteile der Geometrie in die GML-Daten eingeführt und der String mit der GML-Geometrie wird wieder ausgegeben.

```
# GML-Geometrien Ausgabe
def exp_gmlGeometrie(self, tab, geom):
    # Bestandteile der Geometrie für die GML-Daten aus der GeoJSON Filtern
    geom_json = json.loads(geom)
    geom_bestand = self.geom_Best(geom_json)
    gml_geom = ""
    if geom_bestand[0] == "Polygon":
        gml_geom = tab * str("\t") + '<gml:' + geom_bestand[0] + ' srsName="' +
self.epsg + '" gml:id="GML_' + str(uuid.uuid1()) + '">\n'
        gml_geom += (tab+1) * str("\t") + '<gml:exterior>\n'
        gml_geom += (tab+2) * str("\t") + '<gml:LinearRing>\n'
        gml_geom += (tab+2) * str("\t") + '<gml:posList srsDimension="' +
str(geom_bestand[1]) + '" count="' + str(geom_bestand[3]) + '">' + geom_bestand[2]
+ '</gml:posList>\n'
        gml_geom += (tab+2) * str("\t") + '</gml:LinearRing>\n'
```

```

gml_geom += (tab+1) * str("\t") + '</gml:exterior>\n'
# Sind Innenhüllen vorhanden?
if len(geom_bestand[4]) > 0:
    for ior, ior_anz in zip(geom_bestand[4], geom_bestand[5]):
        gml_geom += (tab+1) * str("\t") + '<gml:exterior>\n'
        gml_geom += (tab+2) * str("\t") + '<gml:LinearRing>\n'
        gml_geom += (tab+2) * str("\t") + '<gml:posList
srsDimension="' + str(geom_bestand[1]) + '" count="' + str(ior_anz) + '>' + ior
+ '</gml:posList>\n'
        gml_geom += (tab+2) * str("\t") + '</gml:LinearRing>\n'
        gml_geom += (tab+1) * str("\t") + '</gml:exterior>\n'
    gml_geom += tab * str("\t") + '</gml:' + geom_bestand[0] + '>\n'
return gml_geom

# Bestandteile aus der Geometrie für GML aus GeoJSON ausarbeiten
def geom_Best(self, geom):
    # Bestimmung des Geometrie-Typs
    if geom["type"]=="MultiPolygon":
        geotyp = "Polygon"
    dim = len(geom["coordinates"][0][0][0])
    # Geometrie-Koordinaten Außenhülle
    # Hier könnten auch noch mehrere Polygone vorliegen / Fall wird nicht
    betrachtet
    for poly in geom["coordinates"]:
        for koorList in poly:
            # Länge mit übergeben!!!
            ex_len= len(koorList)
            exterior = ""
            for koor in koorList:
                exterior_koor = ""
                for einzKo in koor:
                    exterior_koor += str(einzKo) + " "
                exterior = exterior_koor + exterior
    # Abfrage der inneren Hüllen
    interior = []
    in_len = []
    if len(poly) > 1:
        i = 1
        while len(poly) > i:
            in_len.append(str(len(poly[i])))
            koor_in = ""
            for koor in poly[i]:
                ior_ko = ""
                for einzKoor in koor:
                    ior_ko += str(einzKoor) + " "
                koor_in = ior_ko + koor_in
            interior.append(koor_in)
            i += 1
    return [geotyp, dim, exterior, ex_len, interior, in_len]

```



Wenn alle Attribute zu einem Objekt betrachtet wurden, sind diese Angaben in XPlanGML-Form als String vorliegend und werden dem String der GML-Daten für den Ausschnitt angehängt. Dies muss für alle Objekte eines Plan-Gebiets durchgeführt werden, bevor dieser im nächsten Schritt als GML-Datei exportiert werden kann. Wie schon zuvor erwähnt, wurde in dieser Arbeit aus zeitlichen Gründen diese Ausgabe der Objekte in XPlanGML-Daten lediglich für die Klassen BP\_Plan, BP\_Bereich, BP\_BaugebietsTeilFlaeche und BP\_StrassenVerkehrsFlaechen umgesetzt.

Für die Erzeugung der fehlenden Funktionen für die weiteren XPlanung-Objekte der Bebauungspläne und der anderen Planwerke müssen die folgenden Punkte des Schemas in den Objekt-Funktionen umgesetzt werden:

1. Abfragen der Attribute aus der Datenbank
2. String-Variable für GML-Daten erzeugen
  - a. Erzeugen der einführenden GML-Daten
3. Funktionen für die Oberklassen
  - a. String erzeugen für GML-Daten
  - b. Erzeugen der Attribute im GML-Format
  - c. Ausgabe des Strings mit den GML-Daten
4. GML-Geometrien erzeugen
5. Eigenen Attribute des Objekts als GML-Daten in String anfügen
6. Ausgabe der GML-Daten zu dem Objekt

Die weiteren fehlenden Klassen müssen noch anhand des dargestellten Schemas und mit Hilfe der bereits vorliegenden Funktionen zu den XPlanung-Objekten umgesetzt werden. Dazu werden als erstes aus allen Tabellen der Datenbank die Attribute der XPlanung-Objekte abgefragt. Bei der Abfrage von Daten aus den Oberklassen können die vorliegenden Funktionen zur Abfrage verwendet werden, da die Abfragen so gestaltet wurden, dass nur die ID der Objekte übergeben werden muss und die Attribute zu dem Objekt ausgegeben werden. Falls diese noch nicht vorliegen, müssen diese wie die anderen Abfragen gestaltet werden. Anschließend muss wieder ein String für die GML-Daten mit den einführenden GML-Daten erzeugt werden. In diesem String werden anschließend zuerst die GML-Daten der Oberklassen von dem Objekt eingetragen. Für diese Oberklassen sind die Funktionen wahrscheinlich schon vorhanden oder müssen mit einer Funktion entwickelt werden und davor die GML-Daten mit

einem String übergeben werden. Für alle bereits realisierten Oberklassen und Klassen ist in dem Anhang eine Objekt-Liste vorhanden. Nachdem die Oberklassen des Objekts in den String der XPlanGML-Daten eingetragen wurden, können die eigenen Attribute des Objekts an den String der Funktion direkt übergeben werden. Dies wird immer direkt in der Funktion des Objekts durchgeführt. Für die GML-Daten von den Geometrien muss die bereits vorhandene Funktion um die noch fehlenden Geometrietyten erweitert werden, denn wie schon zuvor beschrieben, wurde in dieser Arbeit lediglich der Geometrietyt Polygon umgesetzt. Dabei muss sowohl die Funktion zum Auslesen der GeoJSON aus der Datenbank als auch die Erzeugung der GML-Geometriedaten erweitert werden. Abschließend können aus den Funktionen die GML-Daten für die fehlenden XPlanung-Objekte ausgegeben werden.

### Schreiben der GML-Datei

Wie im UML-Diagramm in Abbildung 10 ist der letzte Schritt im Ablauf des Exports, dass die GML-Daten, die bis jetzt nur in einem String vorliegen, in einer GML-Datei gespeichert werden. Die Speicherung der XPlanGML-Daten wird mit dem im Kapitel 3.1.5 vorgestellten IO-Modul über einen Datenstrom in eine Textdatei realisiert. Die Umsetzung dieses Datenstroms ist in dem folgenden Ausschnitt des Programmcodes dargestellt:

```
if newGMLfile != None:
    # Boolean-Werte anpassen für die Validierung
    newGMLfile = newGMLfile.replace('True', 'true')
    newGMLfile = newGMLfile.replace('False', 'false')
    try:
        # Die entsprechende GML-Datei öffnen
        file = open(self.params["datei"], "w" )
        # Schreiben des Inhalts der GML-Datei
        file.write(newGMLfile)
        newGMLfile = self.params["plangebiet"]
        # Schließen der GML-Datei
        file.close()
    except IOError:
        self.tools.showError("Fehler beim Schreiben der GML-Datei!")
```

Als erstes wird bei der Erstellung der XPlanGML-Datei geprüft, ob in der Variablen „newGMLfile“ Daten vorliegen. Wenn dies der Fall ist, werden im Schritt danach alle booleschen Werte aus den GML-Daten ausgetauscht, da in Python die booleschen Werte großgeschrieben werden und in den GML-Daten diese klein geschrieben werden müssen. Mit der Standardfunktion „replace()“ von Python können Bestandteile aus dem String

ausgetauscht werden, wobei ein altes Argument mit einem neuen Argument ersetzt wird (STEYER 2018b, S. 196 f). Anschließend können die XPlanGML-Daten mit den Funktionen des IO-Moduls in eine GML-Datei geschrieben werden. Dabei wird als erstes in dem angegeben Dateipfad aus der GUI die entsprechende Datei mit der Funktion „open“ im schreibenden Modus geöffnet. Mit der Funktion „write“ werden die GML-Daten aus der Variablen „newGMLfile“ in die GML-Datei geschrieben. Abschließend muss dann nur noch über die Funktion „close“ die Datei geschlossen werden und die XPlanGML-Datei wurde erzeugt. Mit dem „try“ und „except“ um den Speicherablauf soll der Schreibvorgang bezüglich Ausnahmebedingungen mit einem IOError abgesichert werden (STEYER 2018a, S. 215).

## 4 Ergebnisse

Für die Darstellung des Ergebnisses aus der XPlanung-Exportschnittstelle muss zuerst in QGIS ein Bebauungsplan mit dem Plugin XPlanung erzeugt werden. Ein Beispiel für die Erstellung eines Bebauungsplans mit dem Plugin in QGIS ist in Abbildung 13 dargestellt. Für das Beispiel wurde ein fiktiver Bebauungsplan für die Gemeinde Spiekeroog erzeugt. Hierbei wurden lediglich die folgenden XPlanung-Objekte verwendet: BP\_Plan, BP\_Bereich, BP\_BaugebietsTeilFlaeche und BP\_StrassenVerkehrsFlaechen. Diese sind auch in der Abbildung 13 unten links im Layerverzeichnis mit den beiden Layern für die Bearbeitung und der Darstellung ausgeführt. Es wurden bewusst in dieser Ergebnisdarstellung nur diese XPlanung-Objekte für den Bebauungsplan verwendet, da auch nur diese Objekte mit der Exportschnittstelle aus der Datenbank exportiert werden können.

Nach der Erstellung des Bebauungsplans kann dieser mit dem prototypischen Export-Tool in eine XPlanGML-Datei exportiert werden. Hierfür müssen wie schon zuvor beschrieben nur das dargestellte Plangebiet, die XPlanung-Version und ein Speicherort für die GML-Datei in der GUI des Export-Tools angegeben werden. Anschließend wird an den gewünschten Speicherort die GML-Datei zu dem Bebauungsplan gespeichert.

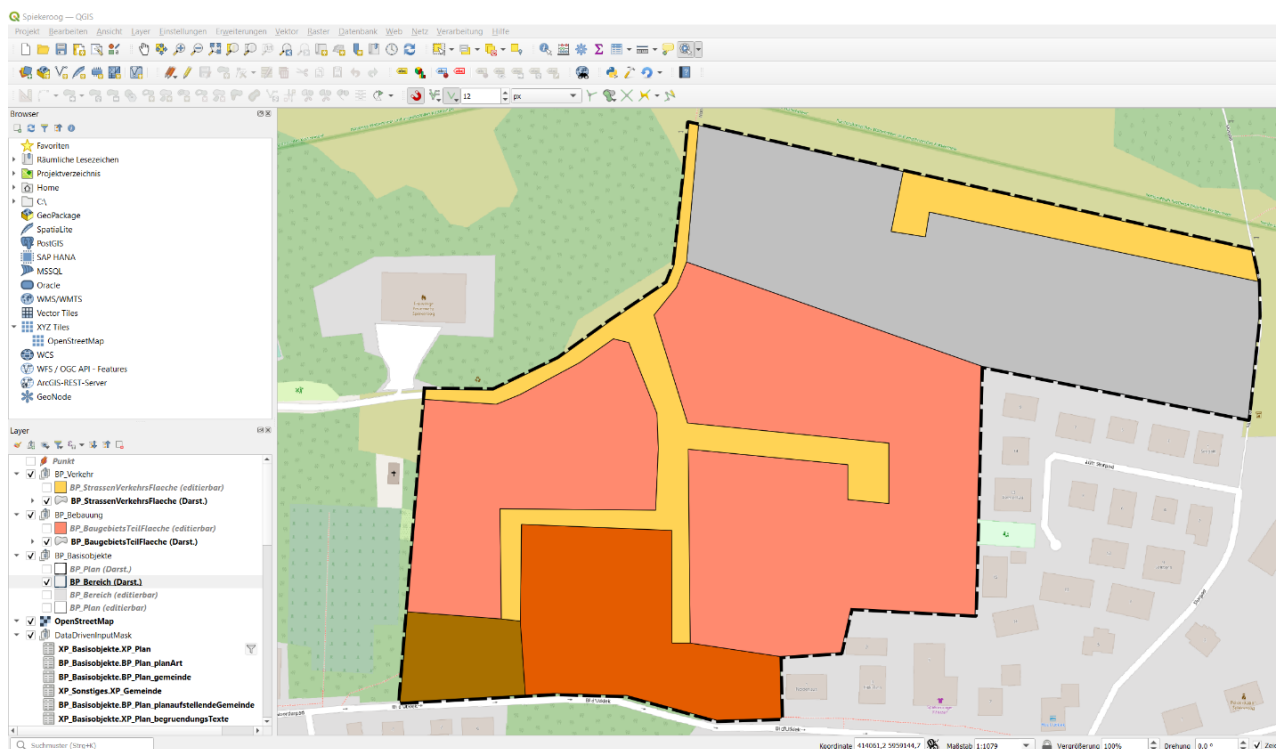


Abbildung 13: Ausschnitt aus QGIS mit dem erzeugten Bebauungsplan mit dem XPlanung-Plugin in Spiekeroog. (eigene)

Ein Ausschnitt des Ergebnisses dieser XPlanGML-Datei des Bebauungsplans ist in Abbildung 14 dargestellt. Die Datei wurde mit dem freien Texteditor Notepad++ geöffnet, da mit diesem XML-Daten erkannt werden. Mit der Software können die Elemente der GML-Daten erkannt und unterschieden werden.

```

<?xml version="1.0" encoding="utf-8"?>
<xplan:XPlanAuszug xmlns:wfs="http://www.opengis.net/wfs" xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:xsi="http:
  <gml:boundedBy>
    <gml:Envelope srsName="EPSG:25832">
      <gml:lowerCorner>413937.509793776 5958912.364513981</gml:lowerCorner>
      <gml:upperCorner>414302.997339045 5959166.465715348</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <gml:featureMember>
    <xplan:BP_Plan gml:id="GML_03d1ba5c-c867-41c6-8052-a50171938a1e">
      <gml:boundedBy>
        <gml:Envelope srsName="EPSG:25832">
          <gml:lowerCorner>413937.509793776 5958912.364513981</gml:lowerCorner>
          <gml:upperCorner>414302.997339045 5959166.465715348</gml:upperCorner>
        </gml:Envelope>
      </gml:boundedBy>
      <xplan:name>Spiekeroog</xplan:name>
      <xplan:nummer>20220809_1001</xplan:nummer>
      <xplan:internalId>1</xplan:internalId>
      <xplan:beschreibung>Alle Bebauungspläne in Spiekeroog.</xplan:beschreibung>
      <xplan:kommentar>Die meisten müssen noch digitalisiert werden</xplan:kommentar>
      <xplan:erstellungsmassstab>20000</xplan:erstellungsmassstab>
      <xplan:bezugshoehe uom="m">13.0</xplan:bezugshoehe>
      <xplan:raeumlicherGeltungsbereich>
        <gml:Polygon srsName="EPSG:25832" gml:id="GML_1dd8bd90-17e0-11ed-b84a-983b8feb0685">
          <gml:exterior>
            <gml:LinearRing>
              <gml:posList srsDimension="2" count="24">414072.283326094 5958912.364513981 414099.694891989
            </gml:LinearRing>
          </gml:exterior>
        </gml:Polygon>
      </xplan:raeumlicherGeltungsbereich>
      <xplan:gemeinde>
        <xplan:XP_Gemeinde>
          <xplan:ags>03462014</xplan:ags>
          <xplan:rs>034620014014</xplan:rs>
          <xplan:gemeindeName>Spiekeroog</xplan:gemeindeName>
          <xplan:ortsteilName>Mitte</xplan:ortsteilName>
        </xplan:XP_Gemeinde>
      </xplan:gemeinde>
      <xplan:planaufstellendeGemeinde>
        <xplan:XP_Gemeinde>
          <xplan:ags>03462014</xplan:ags>
          <xplan:rs>034620014014</xplan:rs>
          <xplan:gemeindeName>Spiekeroog</xplan:gemeindeName>
          <xplan:ortsteilName>Mitte</xplan:ortsteilName>
        </xplan:XP_Gemeinde>
      </xplan:planaufstellendeGemeinde>
      <xplan:plangeber>
        <xplan:XP_Plangeber>
          <xplan:name>Lukas Hermeling</xplan:name>
          <xplan:kennziffer>1996</xplan:kennziffer>
        </xplan:XP_Plangeber>
      </xplan:plangeber>
      <xplan:planArt>1000</xplan:planArt>
      <xplan:verfahren>1000</xplan:verfahren>
      <xplan:rechtsstand>1000</xplan:rechtsstand>
      <xplan:hoehenbezug>NHN</xplan:hoehenbezug>
      <xplan:aenderungenBisDatum>2022-08-09</xplan:aenderungenBisDatum>
      <xplan:aufstellungsbeschlussDatum>2022-08-27</xplan:aufstellungsbeschlussDatum>
      <xplan:veraenderungssperreDatum>2022-08-31</xplan:veraenderungssperreDatum>
      <xplan:veraenderungssperre>false</xplan:veraenderungssperre>
      <xplan:staedtebaulicherVertrag>true</xplan:staedtebaulicherVertrag>
      <xplan:erschliessungsvertrag>false</xplan:erschliessungsvertrag>
      <xplan:durchfuehrungsvertrag>true</xplan:durchfuehrungsvertrag>
      <xplan:gruenordnungsplan>false</xplan:gruenordnungsplan>
      <xplan:bereich xlink:href="#GML_aebb96a9-05b4-4c9d-b392-19cd6687ed77" />
    </xplan:BP_Plan>
  </gml:featureMember>
</gml:featureMember>

```

Abbildung 14: Ausschnitt des XPlanGML-Ergebnisses für den Export des Bebauungsplans von Spiekeroog in Notepad++.  
(eigene)

In dem Ausschnitt der XPlanGML ist zuerst der Header der Datei definiert und anschließend ist das Objekt BP\_Plan mit seinen Attributen dargestellt. Bei den Elementen „xplan:XPlanAuszug“ und „gml:posList“ wurden die Zeilen abgeschnitten, da die Attribute und Koordinaten-Werte den Bildausschnitt aufgrund ihrer Länge unlesbar gemacht hätten. Wie hier zu sehen, wurden beispielhafte Daten für einen Bebauungsplan in der Gemeinde Spiekeroog für die Attribute eingetragen. Die gesamte Ergebnisdatei zu dem Bebauungsplan mit der XPlanGML-Datei ist im Anhang hinterlegt.

Um zu überprüfen, ob diese exportierte XPlanGML-Datei den hohen Ansprüchen der Konformitätsbedingungen von XPlanung entsprechen, muss die GML-Datei mit dem XPlanValidator von KRAUSE (2022a) geprüft werden. Auf der Webseite des XPlanValidators können die XPlanGML-Dateien zur Prüfung hochgeladen werden. Anschließend können die Validierungstypen, die untersucht werden sollen, ausgewählt werden. Hierbei wird zwischen semantischen, geometrischen und syntaktischen Fehlern unterschieden. Die Prüfung nach semantischen und geometrischen Fehlern kann je nach Bedarf abgeschaltet werden, die Syntaktik der GML-Dateien wird jedoch immer kontrolliert.

In Abbildung 15 ist das Ergebnis der Validierung der XPlanGML-Datei für das Beispiel für den Bebauungsplan in Spiekeroog dargestellt. Es ist mit dem grünen „valide“ hinter dem Ergebnis zu sehen, dass die gesamte GML-Datei valide gegenüber den Anforderungen an XPlanGML-Dateien ist. Mit den eingefügten grünen Kästen in Abbildung 15 sind die einzelnen Ergebnisse der Prüfung zu den Validierungstypen markiert. Wenn einer dieser Typen nicht valide wäre, würden unter diesem Validierungstyp die dazugehörigen Fehlermeldungen aufgelistet sein.

**Ergebnisse der Validierung**

### Validierungsbericht

Name: **Spiekeroog\_nord\_neu**  
 XPlan Archivname: **Spiekeroog\_nord\_neu.gml**  
 XPlanGML Version: **5.2**  
 Datum: **09.08.2022 14:37**  
 Ergebnis: **valide**  
 Externe Referenzen:

---

Ergebnis der semantischen Validierung: **valide**

Informationen zu den Regeln:  
 Version: 0.9.19  
 Quelle: <https://bitbucket.org/geowerkstatt-hamburg/xplanung/get/v0.9.19.zip>  
 Zusammenfassung

- 191 Validierungsregeln überprüft
- 0 Validierungsregeln nicht erfüllt
- 191 Validierungsregeln erfüllt ( [anzeigen](#) )

| Regel   | Status | Beschreibung | GML Ids |
|---|--------|--------------|---------|
| Ergebnis der geometrischen Validierung: <b>valide</b> |        |              |         |
| Ergebnis der syntaktischen Validierung: <b>valide</b> |        |              |         |

[Kartenvorschau öffnen](#)

[Zurück zu den Validierungseinstellungen](#)

[Weiteren Plan validieren](#)

Download

HTML Report

PDF Report

XML Report

Geometriefehler

Shapefile

Grafik

[Download](#)

Abbildung 15: Ergebnis der Validierung für die XPlanGML-Datei des Bebauungsplans in Spiekeroog. (KRAUSE 2022a)

Auf der rechten Seite der Abbildung 15 gibt es in dem Feld „Download“ die Möglichkeit, den Validierungsbericht in einem der angegebenen Dateiformate runterzuladen. Mit dem obersten Button „Kartenvorschau öffnen“ auf der rechten Seite wird die Kartenvorschau geöffnet, wie in Abbildung 16 zu sehen ist. Somit ist nachvollziehbar, dass die Geometrien aus der Datei als die richtigen XPlanung-Objekte dargestellt werden. Allerdings können die semantischen Inhalte mit dieser Vorschau nicht betrachtet werden. Dies wird in einer weiteren Prüfung untersucht.

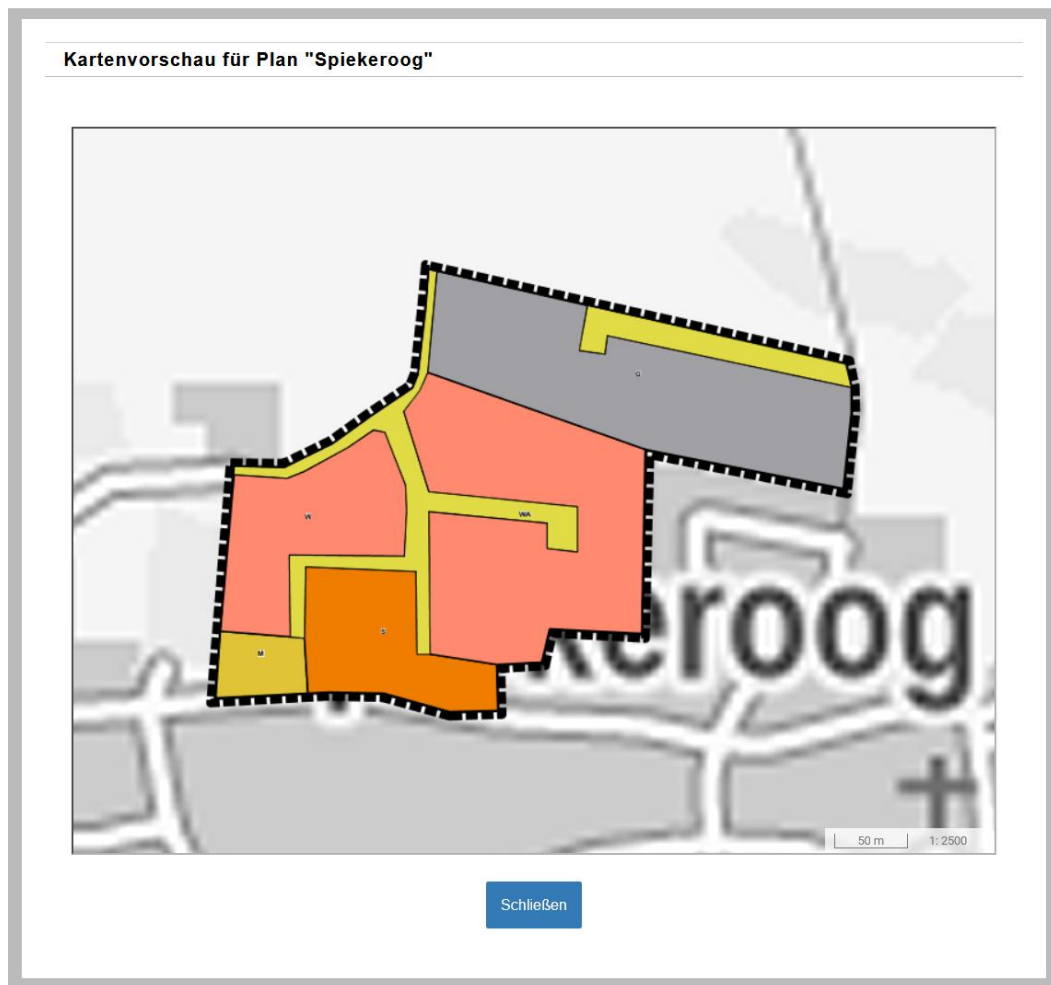


Abbildung 16: Kartenvorschau in dem Validierungsbericht. (KRAUSE 2022a)

Diese weitere Prüfung der exportierten XPlanGML-Datei wurde über den Import der Software ArcGIS Desktop und der Erweiterung GeoOffice XPlanung der Firma VertiGIS geprüft. Bei diesem Import wird untersucht, ob die Datei von anderen XPlanung Softwarelösungen reibungslos importiert werden kann und zusätzlich, ob die semantischen Inhalte der XPlanGML-Datei richtig vorhanden sind.

Wie in Abbildung 17 zu sehen, werden die Geometrien der XPlanung-Objekte ohne Probleme in ArcGIS eingeladen. Zusätzlich sind hinter den Objekten auch die Sachdaten hinterlegt, die mit den Tools von der Erweiterung abgefragt und bearbeitet werden können. Somit ist zu sehen, dass aus der exportierten Datei nicht nur die Geometrien ausgegeben werden, sondern auch die Sachdaten in der exportierten XPlanGML-Datei vorhanden sind sowie richtig importiert und weiterverarbeitet werden können.



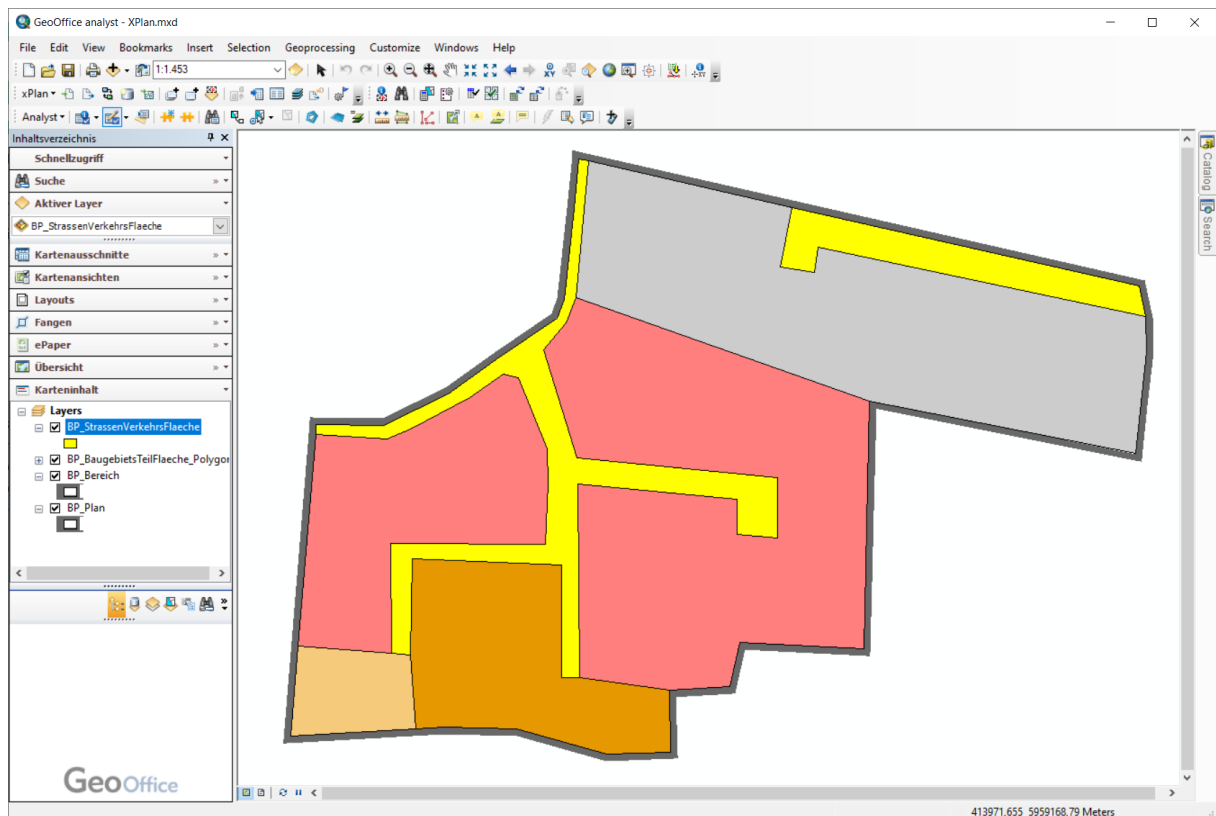


Abbildung 17: Import der Ergebnis-XPlanGML in ArcGIS in das Tool GeoOffice XPlanung. (eigene)

Im Ganzen ist somit ersichtlich, dass mit der Exportschnittstelle aus dieser Arbeit eine konforme XPlanGML-Datei mit den vorgegebenen Objekten erzeugt werden kann.

Bei diesem Ergebnis ist jedoch zu beachten, dass bei der Erzeugung des Ergebnisses bewusst die Fehler bereinigt wurden. Dabei wurden mit Hilfe der Fehlermeldungen des Validators die semantischen und geometrischen Fehler in QGIS behoben, bis ein optimales Ergebnis für eine XPlanGML-Datei vorlag. Dieser Vorgang sollte bei dem Export vermieden werden. Bei einem Export sollten die Daten eigentlich vor dem Export auf semantische und geometrische Fehler kontrolliert werden.

Wie in der Ergebnisdarstellung deutlich wird, ist der Export mit dem Stand der Arbeit für die vorgegeben XPlanung-Objekte möglich. Es müssen allerdings auch noch die weiteren Objekte der BP\_Objekte und für die anderen Planwerke realisiert werden. Dieses und weiteres Ausbaupotential der Exportschnittstelle werden in dem nachfolgenden Kapitel 5 diskutiert.

## 5 Diskussion

In diesem Kapitel werden das Ergebnis sowie die Exportschnittstelle diskutiert. Dabei werden zuerst die erzeugte XPlanGML-Datei und anschließend die Verbesserungsmöglichkeiten für die Exportschnittstelle betrachtet. Bei den möglichen Verbesserungen werden auch die vorgestellten Bibliotheken in Betracht gezogen. Abschließend wird noch ein Vergleich zu der vorhandenen Exportschnittstelle in ArcGIS hergestellt.

Wie in Kapitel 4 zuvor dargestellt wurde, ist der Export von Bebauungsplänen in XPlanGML-Dateien mit der in dieser Arbeit entwickelten Exportschnittstelle möglich. Diese exportierte XPlanGML-Datei ist zudem auch mit dem XPlanValidator auf ihre syntaktische, geometrische und semantische Richtigkeit geprüft worden. Da diese Datei mit den Fehlermeldungen aus dem Validator bereinigt wurde, ist sie auch in den drei Bereichen valide. Aus der Exportschnittstelle wird lediglich eine syntaktisch richtige Datei ausgegeben, wenn bezüglich der XPlanGML-Syntaktik keine Fehler in dem entwickelten Programmcode vorhanden sind. Dies kann im Rahmen der Arbeit nicht endgültig ausgeschlossen werden, da die umgesetzten XPlanung-Objekte sehr komplex sind und somit können sich gerade hinsichtlich der Umsetzung mit Strings schnell Fehler einschleichen. Für die Prüfung hinsichtlich der Validität von Semantik und Geometrie gibt es im Export keine Möglichkeit. Ein solches Tool für die Prüfung der Validität ist ein weiterer Punkt, der in dem Plugin XPlanung fehlt und der nicht nur wegen des Exports, sondern auch für die Nutzung des Plugins XPlanung in QGIS an sich sinnvoll ist. Für die semantischen Inhalte könnten dabei auf der Grundlage der Konformitätsbedingungen die Attribute in der Datenbank kontrolliert werden. Eine weitere Option wäre, direkt bei der Eingabe der Sachdaten mit der Eingabemaske auf Fehler in den semantischen Beziehungen zwischen den Attributen hinzuweisen. Bei der Kontrolle der Geometrien könnte auf schon vorhandene Tools oder Plugins in QGIS zugegriffen werden, wie z. B. das Plugin Geometrieprüfung, und diese vorhandenen Funktionalitäten weiter zu verwenden (QGIS 2019).

Ein weiteres Problem bei dem Ergebnis ist, dass nicht alle Attribute der XPlanung-Objekte in der QGIS Implementierung vorhanden sind. Ein Beispiel dafür ist das Attribut RasterBasis, in dem Rasterdaten der älteren, nicht digital in vektorform vorhandenen Pläne gespeichert werden können. In diesem Attribut wird nur eine Referenz zu dem georeferenzierten Rasterplan gegeben, der den Inhalt des Bereich-Gebiets wiedergibt (KRAUSE 2020, S. 12). Wenn

diese XPlanung-Objekte nicht in den Datenbanken vorhanden sind, können diese natürlich nicht in den XPlanung-Objekten des Plugins erstellt und ebenfalls beim Export nicht berücksichtigt werden.

Der letzte Punkt bezüglich der XPlanGML-Datei ist, dass in dieser Arbeit eine Machbarkeit der Exportschnittstelle analysiert wurde und somit bei der Entwicklung des Exports nur bestimmte XPlanung-Objekte umgesetzt worden sind. Somit fehlen noch einige XPlanung-Objekte für die Bebauungspläne sowie für die anderen Planwerke. Die fehlenden Objekte können anhand des Schemas am Ende des Kapitels 3.2.3 nachträglich entwickelt werden. Nach Umsetzung aller XPlanung-Objekte für die Bebauungspläne und die anderen Planwerke, kann diese veröffentlicht werden. Es wäre auch eine Möglichkeit, die Schnittstelle für die Planwerke einzeln zu veröffentlichen, dabei müssen die einzelnen Planwerke jedoch vollständig vorhanden sein. Mit dem jetzigen Stand kann der Export noch nicht veröffentlicht werden oder nur für die umgesetzten Objekte genutzt werden. Wenn es unvollständig veröffentlicht werden würde, müssten die Benutzer\*innen in der GUI auf dem ersten Blick über die Unvollständigkeit informiert werden, damit es zu keinen Verlusten in den Daten kommt.

Gerade hinsichtlich des programmierten Codes für die Exportschnittstelle gibt es noch einiges Potential zur Verbesserung. So wie es zurzeit realisiert ist, ist es sehr fehleranfällig. Das liegt an der Umsetzung der GML-Daten mit den Strings. Die Daten für die XPlanGML-Daten werden teilweise durch „Copy and Paste“ in den Strings erzeugt und sind somit anfällig für syntaktische Fehler. Diese Fehler müssen aufwendig gesucht werden, indem die XPlanGML-Datei mit dem XPlanValidator geprüft wird. Bei syntaktischen Fehlern wird von den XPlanValidator eine Fehlermeldung mit der Zeile des Fehlers in der Datei ausgegeben. Gerade bei der GML-Syntaktik mit dem Anfangs- und Endtag sowie bei der Angabe von Eigenschaften zu den GML-Attributen können schnell Fehler auftreten.

Diese zuvor beschriebenen Fehler könnten bei der Entwicklung der Exportschnittstelle mit den Bibliotheken verhindert werden. Wenn die lxml-Bibliothek erweitert werden würde für GML-Daten oder auch speziell für die XPlanGML-Daten, könnten einige Fehler in der Syntaktik bei der Entwicklung verhindert werden. In dieser Bibliothek werden Elemente mit den Eigenschaften und den zugehörigen Werten auf einfache Weise angelegt, sodass bei der Ausgabe fehlerfreie XML-Daten mit allen Angaben erzeugt werden. Es werden unter Angabe des Elements automatisch der Anfangs- und Endtag gesetzt. Eine solche Vorlage für die GML-

Struktur würden die Entwicklung der XPlanung-Objekte für die XPlanGML-Daten erheblich erleichtern. Die Weiterentwicklung dieser Bibliothek ist allerdings ebenfalls mit einem sehr hohen Aufwand und der Notwendigkeit fortgeschrittener Programmierkenntnisse verbunden. Doch die Vorteile, die aus der Weiterentwicklung der Bibliothek entspringen, würden für die ganze Entwicklung der Exportschnittstelle von Vorteil sein, da zum einen Fehler im Prozess der Entwicklung reduziert werden würden und die Verwendung der Bibliothek eine Zeitersparnis bei der weiteren Entwicklung der fehlenden XPlanung-Objekte mit sich führen würde. Die Zeitersparnis würde sich nicht nur bei der weiteren Entwicklung, sondern auch bei der Fehlersuche erheblich bemerkbar machen.

Wie in der Durchführung in Kapitel 3.2.3 beschrieben, müssen auch die XPlanGML-Geometrien in dem Code extra mit Funktionen ausgelesen und erzeugt werden. Die pygml-Bibliothek kann leider in diesem Fall nicht verwendet werden, da diese lediglich OGC GML-Geometrien erzeugt. Für die XPlanGML-Daten werden die GML-Geometrien durch die XSD-Datei „gmlProfilexplan.xsd“ weiter eingeschränkt. Diese Einschränkungen können mit der pygml-Bibliothek in dem jetzigen Umfang leider nicht realisiert werden. Auch hierfür wäre es notwendig, die vorhandene Bibliothek für XPlanGML-Geometrien zu erweitern. Dies wäre ebenfalls mit einem hohen Aufwand und entsprechend hohen Programmierkenntnissen verbunden. Es ist sinnvoller, die bereits vorhandene Bibliothek, die auf GML-Geometrien basiert, zu verwenden, um die Umsetzung der XPlanGML-Geometrien zu realisieren, da die XPlanGML-Geometrien aus dem GML-Standard spezifiziert werden. Dies wäre jedoch mit einem gewissen Programmieraufwand verbunden, würde aber ebenfalls eine Vereinfachung für die Weiterentwicklung der Exportschnittstelle sein.

Wenn die Entwicklung ohne die Erweiterungen der Bibliotheken fortgeführt werden sollte, ist dies auch realisierbar nach dem aufgestellten Schema in Kapitel 3.2.3 und der Erzeugung der XPlanGML-Daten über die eigenen erzeugten Strings. Dies ist bei den vielen XPlanung-Objekten mit einem gewissen Zeitaufwand verbunden, aber mit Hilfe des Schemas, der schon vorhandenen Funktionen für die Objekte und der abstrakten Oberklassen, die bereits in Funktionen vorhanden sind, ist die Realisierung einfacher und schneller umsetzbar. Für Entwickler\*innen, die diesen Code nicht kennen, ist es sinnvoll, sich über diese Arbeit in den Programmcode der Exportschnittstelle einzuarbeiten.

Um den Vergleich mit einer schon vorhandenen Exportschnittstelle in einem GIS zu ziehen, kann der Export nach XPlanGML in ArcGIS über die Erweiterung GeoOffice xPlanung verwendet werden. Die GUI für die Exportschnittstelle ist in Abbildung 18 dargestellt. An der oberen rechten Ecke der Export-GUI sind zwei Buttons „Plan aus der Karte“ und „Plan aus der Tabelle“ zu sehen, mit denen die zu exportierenden Plan-Gebiete ausgewählt werden können. Diese ausgewählten Plan-Gebiete werden mit der Gml-Id und den Plannamen in der darunter liegenden Tabelle aufgelistet. Unter der Liste mit den Plan-Gebieten ist zuerst eine Checkbox, mit der man entscheiden kann, ob die Pläne in einzelne GML-Dateien oder in eine XPlanGML-Datei gespeichert werden sollen. Wenn die Pläne einzeln abgespeichert werden sollen, muss nur ein Ordner-Pfad angegeben werden, in dem die einzelnen Pläne mit dem Plannamen als GML gespeichert werden. Wenn dies nicht der Fall ist, wird – wie in der Abbildung zu sehen – der ganze GML-Dateipfad angegeben. Mittig über dem Dateipfad in der GUI kann das Zielformat mit einer Dropdown-Liste angegeben werden. Hierbei kann man zwischen den XPlanung-Versionen 4.0, 4.1, 5.0, 5.1 und 5.2 entscheiden und die Auswahl ist automatisch immer auf die aktuellste Version 5.2 eingestellt. Unter dem Textfeld für den Dateipfad können noch drei weitere Optionen ausgewählt werden. Mit der ersten Option kann entschieden werden, ob die Pläne vor dem Export geprüft werden sollen. Bei der zweiten Checkbox wird abgefragt, ob nur XPlan-konforme Pläne exportiert werden dürfen. Und bei der dritten Option kann ausgewählt werden, ob Präsentationsobjekte mit ungültigen Art-Verweis exportiert werden sollen. (VERTIGIS GMBH 2022)

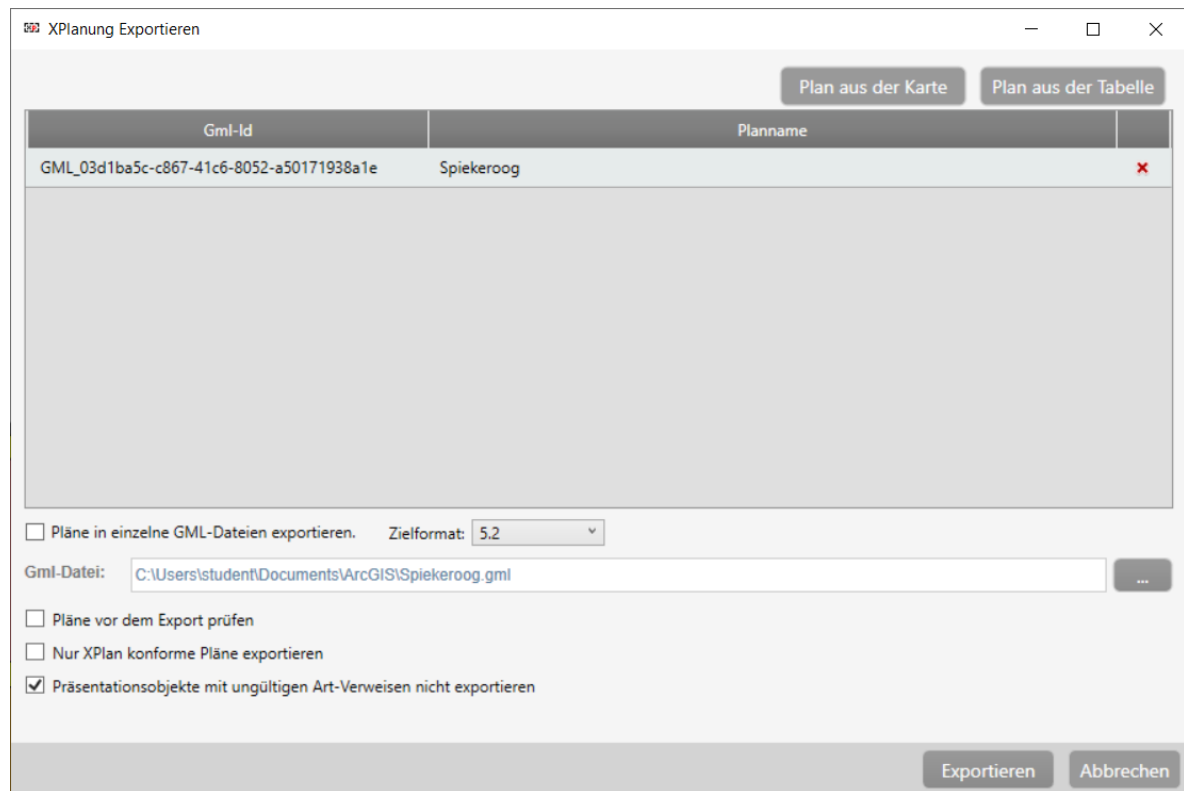


Abbildung 18: Exportschnittstelle für XPlanGML-Dateien in ArcGIS aus der Erweiterung GeoOffice xPlanung. (eigene)

Wie in Abbildung 18 zu sehen, gibt es in der Exportschnittstelle von ArcGIS eine Vielfalt von Optionen, die bei dem Export von XPlanGML-Dateien berücksichtigt werden können. Gleiche Optionen wie bei der Exportschnittstelle von QGIS sind, dass die Pläne aus einer Tabelle ausgewählt werden können, die zu exportierende XPlanung-Version angegeben werden kann und der Dateipfad in dem Texteingabefeld eingegeben oder über den Button „...“ mit einem Dialogfenster zu der Ordnerstruktur angegeben werden kann. Somit wird deutlich, dass grundlegende Funktionen in beiden Exportschnittstellen ähnlich umgesetzt wurden. Natürlich wurden in dem QGIS Tool noch nicht alle Funktionalitäten vollständig umgesetzt, aber die ähnlichen Ansätze in den Exporten nach XPlanGML sind erkennbar. Die Exportschnittstelle von ArcGIS ist mit den ganzen Optionen eine gelungene Lösung, die mit einigen Extras bestückt wurde. Diese Extras sollten für die Exportschnittstelle in QGIS erst einmal hintenangestellt werden. Bei der Entwicklung sollte der Fokus zuerst auf die Grundlagen des Exports liegen, sodass dieser vollständig für alle XPlanung-Objekte durchgeführt werden kann. Erst nach der Entwicklung eines vollständigen Exports für eine XPlanung-Version kann anschließend über weitere Funktionalität nachgedacht werden. Dabei sollten die Funktionen für die semantische und geometrische Prüfung im Anschluss realisiert werden, da die Prüfung der Validität die wichtigste noch fehlende Prüfung im Zusammenhang mit dem Export ist.

Es zeigt sich, dass noch viel Arbeit bei der weiteren Entwicklung der Exportschnittstelle sowie den weiteren Funktionen zu dem Plugin in QGIS bevorsteht. Hierbei kann an verschiedenen Stellen angesetzt werden, um die Entwicklung der Exportschnittstelle zu erleichtern und auch die Entwicklung von der Erzeugung von GML-Daten voranzubringen. Dabei ist es auch ein wichtiger Punkt, die Weiterentwicklung der Bibliotheken zu fördern. Wenn aber nur die Exportschnittstelle nach XPlanGML in QGIS entwickelt werden soll, kann anhand dieser Arbeit die Weiterentwicklung vorangetrieben werden. Ein Ausblick der Funktionalitäten, die noch realisiert werden können, ist mit dem Vergleich des Exports in ArcGIS gegeben worden.

Im Hinblick auf die Entwicklungen im BIM-Bereich, ist der Datenaustauschstandard XPlanGML ein sehr wichtiger Bestandteil der Planung in Deutschland. BIM-Modelle werden in immer mehr Bereichen eingesetzt, so auch bei den Bauanträgen. Dabei ist das Projekt mit dem Titel „Konzept für die nahtlose Integration von Building Information Modeling (BIM) in das behördliche Bauantragsverfahren“ (Kurztitel „BIM-basierter Bauantrag“) ein Forschungsprojekt, bei dem der Einsatz von BIM-Modellen in bauordnungsrechtlichen Verwaltungsverfahren analysiert wurden, gerade hinsichtlich des Bauantragsverfahren. Aus diesem Grund wurde auch die Kombination von BIM-Datenformaten mit den deutschen Standards XPlanung und XBau untersucht. Dabei wurde betrachtet, welche Informationen aus den digitalen Bebauungsplänen für das BIM genutzt werden können und ebenfalls welche BIM-Information von dem Bebauungsplan entgegengenommen werden können. (ZUKUNFT BAU 2020a, S. 4 ff)

In Rahmen dieses Projektes wurde eine prototypische Software entwickelt, in der sowohl XPlanGML-Dateien für die Bebauungspläne als auch BIM-Datenmodelle eingeladen werden können. Diese Software des Projektes ist in Abbildung 19 dargestellt, wobei man in der Mitte eine Visualisierung der XPlanGML-Daten zum Bebauungsplan und das darauf abgebildete BIM-Modell sehen kann. In den anliegenden Fenstern daneben sind die Eigenschaften und Attribute zu dem BIM-Modell sowie zu den XPlanung- und XBau-Daten dargestellt. (ZUKUNFT BAU 2020a, S. 32 ff)

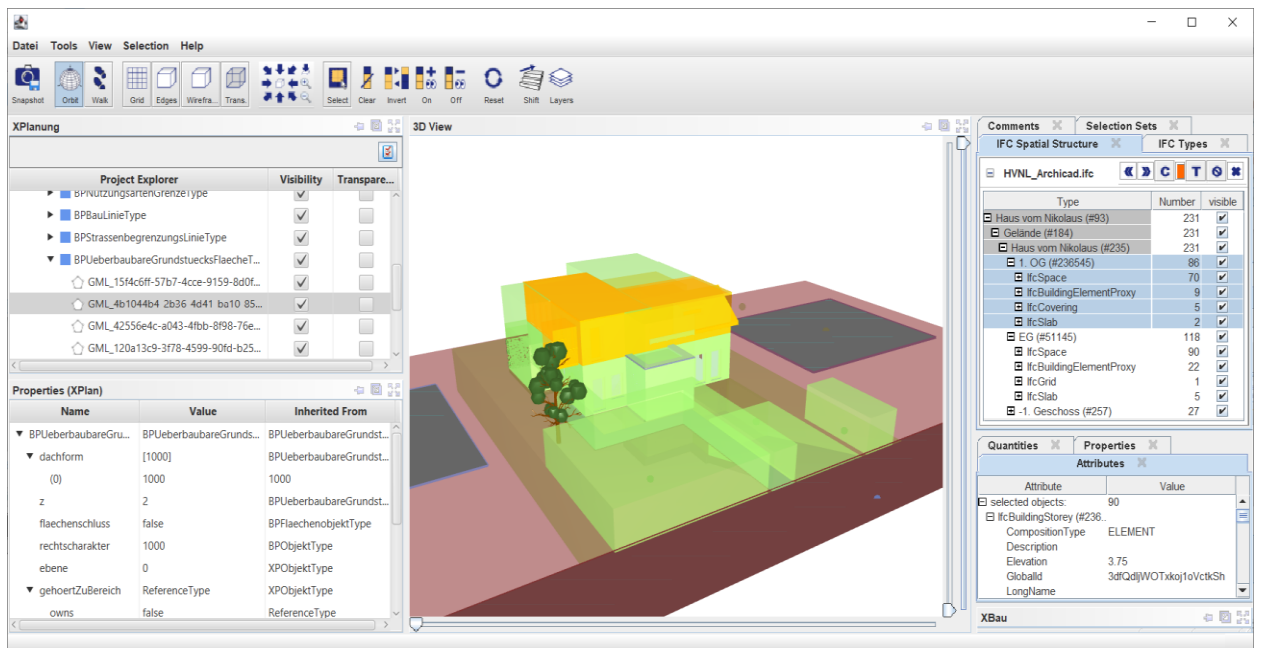


Abbildung 19: Integration und Visualisierung der antragsrelevanten Daten für einen Bauantrag in der prototypischen Software. (ZUKUNFT BAU 2020b, S. 8)

Somit lässt sich herausstellen, dass nicht nur die reine Erstellung des Bebauungsplans mit dem Standard XPlanung wichtig ist, sondern der Export der XPlanGML-Dateien immer mehr an Relevanz gewinnt, da diese Daten mit dem Standardformat in Prozessen wie dem BIM-basierten Bauanträgen miteinbezogen und die Planungsprozesse somit effizienter gestaltet werden.



## 6 Fazit und Ausblick

Bei der Erzeugung von Bebauungsplänen werden bei den Kommunen immer mehr GIS-Schalen mit dem Standard XPlanung eingesetzt, da der Standard nach dem Beschluss des IT-Planungsrates bis 2022 in allen deutschen Kommunen für alle Planwerke umgesetzt werden muss (BILL 2010, S. 673; IT-PLANUNGSRAT 2017). Gemeinden mit einem geringen Budget sind auf Open Source Produkte angewiesen, um mit den Entwicklungen im Planungsbereich mitgehen zu können.

Somit wurde in dieser Arbeit die Erweiterung des Plugins XPlanung der Open Source Software QGIS um eine Exportschnittstelle für das objektorientierte Datenaustauschformat XPlanGML untersucht. Dabei wurde für bestimmte XPlanung-Objekte der Bebauungspläne ein prototypischer Export entwickelt. Wie in dem Ergebnis der Machbarkeitsanalyse zu sehen ist, wurde die Exportschnittstelle so weit entwickelt, dass die entsprechenden XPlanung-Objekte aus dem Plugin in einer XPlanGML-Datei gespeichert werden konnten. Dennoch ist diese für die Veröffentlichung der Exportschnittstelle nicht ausreichend, da einige XPlanung-Objekte im Export noch nicht realisiert wurden. Dafür wurde aus der prototypischen Umsetzung noch ein Schema abgeleitet, mit dem sich eine weitere Entwicklung realisieren lässt. Gerade die Erweiterungen der vorgestellten Bibliotheken sind Ansätze, die Entwickler\*innen aufnehmen können, um eine verbesserte Entwicklung mit GML-Daten zu erreichen. Diese Weiterentwicklungen in den Bibliotheken würden die Entwicklung der Exportschnittstelle wie schon beschrieben erheblich vereinfachen. Der zurzeit gewählte Ansatz zur Umsetzung des Exports birgt eine gewisse Fehleranfälligkeit, da auf der String-Grundlage gearbeitet wird, wobei der/die Entwickler\*in die GML-Daten per Hand erzeugt und sich somit schnell und einfach Fehler einschleichen können. Diese Fehler könnten ebenfalls mit den vorgeschlagenen Weiterentwicklungen der Bibliotheken aus dem Kapitel 5 eingeschränkt werden.

Für die weitere Entwicklung der Exportschnittstelle soll der erzeugte Programmcode veröffentlicht werden. Hierfür bietet sich die Internet-Plattform GitHub an, da auch hier bereits der vorhandene Programmcode des Plugins XPlanung von STRÖBL (2021a) veröffentlicht wurde. Die Entwicklungen zu der Exportschnittstelle könnten hier zusätzlich mit der Arbeit abgelegt werden, damit interessierte Entwickler\*innen das Plugin an dieser Stelle wiederfinden und die Möglichkeiten zur Weiterentwicklung des Exports erhalten. Somit kann auf die Entwicklung in dieser Arbeit aufgebaut und der XPlanGML-Export realisiert werden.

Der Export von XPlanGML-Dateien ist ein essenzielles Werkzeug, nicht nur um die Pläne nach dem Beschluss für die Öffentlichkeit zugänglich zu machen, sondern auch um die Beteiligten im Planungsprozess mit einzubinden. Hierfür muss die XPlanGML-Datei zu der Planung auf einem Server zur Verfügung gestellt werden und die Beteiligten bekommen auf die Datei verschiedene Zugriffsrechte, um an der Planung mitzuwirken. Somit ist es für den Planungsprozess wichtig, dass die Daten aus der Software exportiert und weiterverarbeitet werden können.

Zusätzlich wird der Export der XPlanGML-Dateien auch im weiteren Planungsprozessen, wie dem digitalen Bauantrag benötigt. Die Verwendung der exportierten XPlanGML-Datei ist in der Diskussion und der Abbildung 19 zu dem Projekt „Digitaler Bauantrag“ dargestellt. Dabei werden die Daten des Bebauungsplans mit dem BIM-Modell abgeglichen und Information aus den Modellen rausgezogen. Somit können mit dieser Software vor dem Einreichen eines Bauantrags schon Prüfungen durchgeführt und Probleme im Planungsprozess aufgedeckt werden. Dies ist nur ein Prozess, in dem die standardisierten Pläne für einen effizienten Ablauf genutzt werden können. Für den Standard der Planwerke gibt es noch viele weitere Anwendungsbereiche, in denen über das Standardaustauschformat XPlanGML die Effizienz im Planungsprozess gesteigert werden kann.

Aus diesen dargestellten Gründen ist es wichtig, dass in der Software QGIS in dem Plugin XPlanung die Exportschnittstelle mit dem hier dargestellten Schema und Ansätzen realisiert wird und in den Gemeinden und Städten mit geringeren Budgets diese Open Source Lösung für den Standard genutzt werden kann. Aber nicht nur das Export-Tool sollte realisiert werden, sondern auch noch die weiteren Tools, die in dieser Arbeit genannt wurden, und auf der ToDo-Liste des Plugins stehen. Dadurch würden die kleineren Gemeinden und Städte bei der Weiterentwicklung dieser Planungsprozess nicht auf der Strecke bleiben, da sie mit einem voll entwickelten Open Source Produkten gleich gut aufgestellt sind wie mit den kommerziellen Produkten.

## Literaturverzeichnis

BARTELME, NORBERT (2005): Geoinformatik. Modelle. Strukturen. Funktionen. 4., vollst. überarb. Aufl. Berlin [u.a.]: Springer.

BENNER, JOACHIM (2020a): Konformitätsbedingungen XPlanung 5.2.1. Version 7.4.2. Online verfügbar unter <https://xleitstelle.de/filebrowser/download/890>, zuletzt geprüft am 11.07.2022.

BENNER, JOACHIM (2020b): XPlanung Struktur und Konzepte. Version 5.3. Online verfügbar unter <https://xleitstelle.de/filebrowser/download/819>, zuletzt geprüft am 14.04.2022.

BENNER, JOACHIM; KÖPPEN, ANTJE; KLEINSCHMIT, BIRGIT; KRAUSE, KAI-UWE; WICKEL, MARTIN (2008): XPlanung – Neue Standards in der Bauleit- und Landschaftsplanung. In: *Digital Design in Landscape Architecture*, S. 240–248. Online verfügbar unter [https://www.researchgate.net/profile/birgit-kleinschmit/publication/267990165\\_xplanung\\_-\\_neue\\_standards\\_in\\_der\\_bauleit-und\\_landschaftsplanung](https://www.researchgate.net/profile/birgit-kleinschmit/publication/267990165_xplanung_-_neue_standards_in_der_bauleit-und_landschaftsplanung).

BENNER, JOACHIM; KRAUSE, KAI-UWE (2006): XPlanung–Der Standard in der Bauleitplanung. Online verfügbar unter <https://gispoint.de/artikelarchiv/gis/2006/gis-ausgabe-112006/2626-xplanung-der-standard-in-der-bauleitplanung-austausch-standard-xplan-gml.html>, zuletzt geprüft am 30.06.2022.

BENNER, JOACHIM; KRAUSE, KAI-UWE; MÜLLER, MARKUS U. (2005): Elektronische Planzeichenverordnung–Modellierung, Datenaustausch und Visualisierung von Bauleitplänen mit OGC-Standards. In: Manfred Schrenk (Hg.): CORP 2005. Proceedings of 10th symposion on "Information- and communication technologies (ICT) in urban planning and spatial development and impacts of ICT on physical space". February 22 - February 25, 2005, Vienna University of Technology = Beiträge zum 10. Symposion zur Rolle der Informations- und Kommunikationstechnologie in der Stadtplanung und Regionalentwicklung sowie zu den Wechselwirkungen zwischen realem und virtuellem Raum. Wien: Selbstverl. des Instituts für EDV-Gestützte Methoden in Architektur und Raumplanung. Online verfügbar unter [https://conference.corp.at/archive/corp2005\\_krause.pdf](https://conference.corp.at/archive/corp2005_krause.pdf).

BILL, RALF (2010): Grundlagen der Geoinformationssysteme. 5., völlig neu bearb. Auflage 2009. Berlin: Wichmann.

BRINKHOFF, THOMAS (2013): Geodatenbanksysteme in Theorie und Praxis. Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial. 3., überarbeitete und erweiterte Auflage. Berlin [u.a.]: Wichmann. Online verfügbar unter [http://www.content-select.com/index.php?id=bib\\_view&ean=9783879075720](http://www.content-select.com/index.php?id=bib_view&ean=9783879075720).

DE LANGE, NORBERT (2020): Geoinformatik in Theorie und Praxis. Grundlagen von Geoinformationssystemen, Fernerkundung und digitaler Bildverarbeitung. 4., überarb. u. akt. Auflage 2020. Berlin, Heidelberg: Springer Berlin Heidelberg.

DUAN, XINXIN (2022): Aufbau XPlanGML, 20.05.2022. E-Mail an Lukas Hermeling.

GOLL, JOACHIM (2014): Architektur- und Entwurfsmuster der Softwaretechnik. Mit lauffähigen Beispielen in Java. 2., aktualisierte Aufl. Wiesbaden: Springer Vieweg.

IT-PLANUNGSRAT (2017): Beschluss 2017/23 - Standardisierungsagenda: Austausch im Bau- und Planungsbereich | IT-Planungsrat. Online verfügbar unter <https://www.it-planungsrat.de/beschluss/beschluss-2017-23>, zuletzt geprüft am 30.06.2022.

KRAUSE, KAI-UWE (2020): Objektartenkatalog 5.2 | XLeitstelle. Hg. v. Leitstelle XPlanung / XBau. Online verfügbar unter <https://xleitstelle.de/downloads/xplanung/releases/XPlanung%20Version%205.2.1/Objektartenkatalog.pdf>, zuletzt aktualisiert am 23.02.2020, zuletzt geprüft am 12.07.2022.

KRAUSE, KAI-UWE (2022a): XPlanValidatorWeb. Hg. v. Leitstelle XPlanung / XBau. Online verfügbar unter <https://www.xplanungsplattform.de/xplan-validator/>, zuletzt aktualisiert am 27.06.2022, zuletzt geprüft am 12.07.2022.

KRAUSE, KAI-UWE (2022b): XPlanung | XLeitstelle. Hg. v. Leitstelle XPlanung / XBau. Online verfügbar unter <https://xleitstelle.de/index.php/xplanung>, zuletzt aktualisiert am 25.07.2022, zuletzt geprüft am 25.07.2022.

KRAUSE, KAI-UWE; KRÄTSCHER, ROBERT; BENNER, JOACHIM (2016): Stand der Weiterentwicklung und Umsetzung des Standards XPlanung in Deutschland. In: Manfred Schrenk, Vasily V. Popovich, Peter Zeile, Pietro Elisei und Clemens Beyer (Hg.): REAL CORP 2016. Smart Me Up! How to become and how to stay a Smart City, and does this improve quality of life? : proceedings of

21st international conference on Urban Planning, Regional Development and Information Society = Beiträge zur 21. internationalen Konferenz zu Stadtplanung, Regionalentwicklung und Informationsgesellschaft ; Tagungsband. Wien: CORP - Competence Center of Urban and Regional Planning. Online verfügbar unter <https://core.ac.uk/download/pdf/55284746.pdf>.

OSGEO LIVE (2021): GDAL/OGR — OSGeoLive 14.0 Documentation. Online verfügbar unter [https://live.osgeo.org/de/overview/gdal\\_overview.html](https://live.osgeo.org/de/overview/gdal_overview.html), zuletzt aktualisiert am 06.11.2021, zuletzt geprüft am 12.07.2022.

POSTGIS DEVELOPMENT GROUP (2022a): ST\_AsGeoJSON. Online verfügbar unter [https://postgis.net/docs/ST\\_AsGeoJSON.html](https://postgis.net/docs/ST_AsGeoJSON.html), zuletzt aktualisiert am 26.07.2022, zuletzt geprüft am 03.08.2022.

POSTGIS DEVELOPMENT GROUP (2022b): ST\_AsGML. Online verfügbar unter [https://postgis.net/docs/ST\\_AsGML.html](https://postgis.net/docs/ST_AsGML.html), zuletzt aktualisiert am 26.07.2022, zuletzt geprüft am 03.08.2022.

PYTHON SOFTWARE FOUNDATION (2022a): io. Core tools for working with streams. Online verfügbar unter <https://docs.python.org/3/library/io.html>, zuletzt geprüft am 18.07.2022.

PYTHON SOFTWARE FOUNDATION (2022b): json — JSON encoder and decoder. Online verfügbar unter <https://docs.python.org/3/library/json.html>, zuletzt aktualisiert am 03.08.2022, zuletzt geprüft am 03.08.2022.

PYTHON SOFTWARE FOUNDATION (2022c): uuid. UUID objects according to RFC 4122. Python 3.10.7 documentation. Online verfügbar unter <https://docs.python.org/3/library/uuid.html>, zuletzt aktualisiert am 07.09.2022, zuletzt geprüft am 07.09.2022.

PYTHON SOFTWARE FOUNDATION (2022d): xml.etree.ElementTree. The ElementTree XML API - Python 3.10.5 documentation. Online verfügbar unter <https://docs.python.org/3/library/xml.etree.elementtree.html>, zuletzt geprüft am 14.07.2022.

QGIS (2019): Geometrieprüfung Plugin. Online verfügbar unter [https://docs.qgis.org/2.18/de/docs/user\\_manual/plugins/plugins\\_geometry\\_checker.html](https://docs.qgis.org/2.18/de/docs/user_manual/plugins/plugins_geometry_checker.html), zuletzt aktualisiert am 08.04.2019, zuletzt geprüft am 12.09.2022.

QGIS (2022a): Dokumentation für QGIS 3.16 — QGIS Documentation Dokumentation. Online verfügbar unter <https://docs.qgis.org/3.16/de/docs/index.html>, zuletzt aktualisiert am 02.04.2022, zuletzt geprüft am 04.07.2022.

QGIS (2022b): QGIS Plugins. Online verfügbar unter <https://plugins.qgis.org/>, zuletzt aktualisiert am 04.07.2022, zuletzt geprüft am 04.07.2022.

QGIS (2022c): Willkommen beim QGIS Projekt! Online verfügbar unter <https://www.qgis.org/de/site/>, zuletzt aktualisiert am 04.07.2022, zuletzt geprüft am 04.07.2022.

QT COMPANY (2020): QSqlQuery — Qt for Python. Online verfügbar unter <https://doc.qt.io/qtforpython-5/PySide2/QtSql/QtSqlQuery.html>, zuletzt aktualisiert am 20.11.2020, zuletzt geprüft am 03.08.2022.

QT COMPANY (2022): Qt Designer Manual. Online verfügbar unter <https://doc.qt.io/qt-5/qtdesigner-manual.html>, zuletzt aktualisiert am 07.06.2022, zuletzt geprüft am 04.07.2022.

RICHTER, STEPHAN (2022): lxml - Processing XML and HTML with Python. Online verfügbar unter <https://lxml.de/>, zuletzt aktualisiert am 01.07.2022, zuletzt geprüft am 13.07.2022.

SCHINDLER, FABIAN; KRALIDIS, TOM (2022): geopython/pygml. Hg. v. GitHub. Online verfügbar unter <https://github.com/geopython/pygml>, zuletzt aktualisiert am 15.07.2022, zuletzt geprüft am 15.07.2022.

STEYER, RALPH (2018a): Datei-, Datenträger- und Datenbankzugriffe – Dauerhafte Daten. In: Ralph Steyer (Hg.): Programmierung in Python. Wiesbaden: Springer Fachmedien Wiesbaden, S. 211–223.

STEYER, RALPH (2018b): String-Verarbeitung in Python – Programmierte Textverarbeitung. In: Ralph Steyer (Hg.): Programmierung in Python. Wiesbaden: Springer Fachmedien Wiesbaden, S. 195–209.

STRÖBL, BERNHARD (2016): XPlanung für einen Flächennutzungsplan mit PostGIS und QGIS.

STRÖBL, BERNHARD (2018): XPlanung 5.0 in QGIS: Chaos Computer Club e.V.

STRÖBL, BERNHARD (2019): DataDrivenInputMask Wiki. Hg. v. GitHub. Online verfügbar unter <https://github.com/bstroebel/DataDrivenInputMask/wiki>, zuletzt aktualisiert am 05.07.2022, zuletzt geprüft am 05.07.2022.

STRÖBL, BERNHARD (2020): xplanplugin Wiki. Hg. v. GitHub. Online verfügbar unter <https://github.com/bstroebel/xplanplugin/wiki>, zuletzt aktualisiert am 25.05.2020, zuletzt geprüft am 16.08.2021.

STRÖBL, BERNHARD (2021a): xplanplugin. QGIS Plugin for German standard XPlanung. Hg. v. GitHub. Online verfügbar unter <https://github.com/bstroebel/xplanplugin>, zuletzt aktualisiert am 29.01.2021, zuletzt geprüft am 05.07.2022.

STRÖBL, BERNHARD (2021b): xplanPostGIS. Implementation of German standard XPlanung for PostgreSQL/PostGIS. Hg. v. GitHub. Online verfügbar unter <https://github.com/bstroebel/xplanPostGIS>, zuletzt aktualisiert am 01.12.2021, zuletzt geprüft am 01.12.2021.

THE QT COMPANY (2022): QFileDialog. Qt for Python. Online verfügbar unter <https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QFileDialog.html#more>, zuletzt aktualisiert am 20.11.2020, zuletzt geprüft am 27.07.2022.

UJAVAL GANDHI (2021): Ein Python Plugin erstellen (QGIS3) — QGIS Tutorials and Tips. Online verfügbar unter [http://www.qgistutorials.com/de/docs/3/building\\_a\\_python\\_plugin.html](http://www.qgistutorials.com/de/docs/3/building_a_python_plugin.html), zuletzt aktualisiert am 04.07.2022, zuletzt geprüft am 04.07.2022.

VERTIGIS GMBH (2022): GeoOffice xPlanung. Produktübersicht. Online verfügbar unter <https://resources.geooffice.vertigis.com/documentation/DE/xplanung/index.html>, zuletzt geprüft am 25.07.2022.

WARMERDAM, FRANK; ROUAULT, EVEN (2022): GDAL Documentation. Online verfügbar unter <https://gdal.org/gdal.pdf>, zuletzt geprüft am 12.07.2022.

WELLNHOFER, NICK (2022a): libxml2. Wiki · GNOME. Hg. v. GitLab. Online verfügbar unter <https://gitlab.gnome.org/GNOME/libxml2/-/wikis/home>, zuletzt geprüft am 14.07.2022.

WELLNHOFER, NICK (2022b): libxslt. Wiki · GNOME. Hg. v. GitLab. Online verfügbar unter <https://gitlab.gnome.org/GNOME/libxslt/-/wikis/home>, zuletzt geprüft am 14.07.2022.

WETRANSFORM GMBH (2020): INSPIRE Transformation mit hale»studio (deutsch). Weitere Beteiligte: Thorsten Reitz und Christopher Hönn. YouTube. Online verfügbar unter <https://www.youtube.com/watch?v=3JqiUNOHMdQ>, zuletzt geprüft am 06.07.2022.

WETRANSFORM GMBH (2022a): hale studio. fast, interactive Data Transformation. Unter Mitarbeit von Simon Templer. Online verfügbar unter <https://www.wetransform.to/products/halestudio/>, zuletzt aktualisiert am 06.07.2022, zuletzt geprüft am 06.07.2022.

WETRANSFORM GMBH (2022b): Workshops and Trainings. Online verfügbar unter <https://www.wetransform.to/services/workshops/>, zuletzt aktualisiert am 08.07.2022, zuletzt geprüft am 11.07.2022.

WÜRRIEHAUSEN, FALK; MÜLLER, HARTMUT (2012): Mit XPlanung zu eGovernment 2.0. In: Josef Strobl, Thomas Blaschke und Gerald Griesebner (Hg.): Angewandte Geoinformatik 2012. Beiträge zum 24. AGIT-Symposium Salzburg ; [4. bis 6. Juli 2012 ; agit2012 - GI Impulse vernetzen. AGIT-Symposium. Berlin, Offenbach: Wichmann.

Zukunft Bau (Hg.) (2020a): Abschlussbericht. Konzept für die nahtlose Integration von Building Information Modeling (BIM) in das behördliche Bauantragsverfahren. Ruhr-Universität Bochum. Online verfügbar unter [https://bim-bauantrag.blogs.ruhr-uni-bochum.de/wp-content/uploads/2020/09/Abschlussbericht\\_2020-09-14\\_rev1.pdf](https://bim-bauantrag.blogs.ruhr-uni-bochum.de/wp-content/uploads/2020/09/Abschlussbericht_2020-09-14_rev1.pdf), zuletzt geprüft am 29.08.2022.

Zukunft Bau (Hg.) (2020b): BIM-basierter Bauantrag. Prototypische Software. Ruhr-Universität Bochum. Online verfügbar unter [https://bim-bauantrag.blogs.ruhr-uni-bochum.de/wp-content/uploads/2020/04/003\\_Prototypische\\_Software.pdf](https://bim-bauantrag.blogs.ruhr-uni-bochum.de/wp-content/uploads/2020/04/003_Prototypische_Software.pdf), zuletzt aktualisiert am 04.03.2020, zuletzt geprüft am 29.08.2022.



## Anhang

Der Programmcode des gesamten Plugin XPlanung befindet sich auf der beigefügten CD/USB-Stick im Ordner „Plugin Code“. Im Rahmen dieser Arbeit würden die folgenden Zeilen Programmcode in den Python-Skripten für den Export nach XPlanGML hinzugefügt:

- XPlan.py → Zeile 562 bis 597
- XPlanDialog.py → Zeile 1126 bis 1415
- XPTools.py → Zeile 61 bis 72
- Gesamtes Skript XPExport.py

Für die GUIs wurde mit der Software Qt Designer die die folgenden Ui-Dateien erzeugt:

- Ui\_Export.ui
- Ui\_Export\_Plan\_Bereichsauswahl.ui

Zusätzlich findet sich im Anhang die Mail von der XLeitstelle von Xinxin Duan im Ordner „Mail“ sowie in dem Ordner „XPlanung Version 5.2.1“ die Dateien zu den Spezifikationen der XPlanung-Version 5.2.1.

Die Ergebnisse des Exports sowie des XPlanValidators befinden sich in dem Ordner „Ergebnis“.

## **XPlanung-Objekte Liste**

In dieser Liste werden alle umgesetzten Ober-/Klassen aufgezählt die mit Abfragen an die Datenbank-Tabellen und der Umsetzung in das XPlanGML-Format aufgelistet.

### **Oberklassen:**

- XP\_Plan
- XP\_Bereich
- XP\_Objekt
- XP\_ExterneReferenz
- XP\_Gemeinde
- XP\_Plangeber
- XP\_VerbundenerPlan
- XP\_Hoehenangabe
- XP\_BegrueundungAbschnitt
- XP\_WirksamkeitBedingung
- XP\_VerfahrensMerkmal
- XP\_TextAbschnitt
- XP\_BegrueundungAbschnitt
- XP\_SpezExterneReferenz
- XP\_SPEMassnahmenDaten
- XP\_AbstraktesPraesentationsobjekt

### **Klassen:**

- BP\_Plan
- BP\_Bereich
- BP\_Objekt
- BP\_Dachgestaltung
- BP\_Flaechenobjekt
- BP\_TextAbschnitt
- BP\_AusgleichsFlaeche
- BP\_AnpflanzungBindungErhaltung
- BP\_SchutzPflegeEntwicklungsMassnahme

- BP\_SchutzPflegeEntwicklungsFlaeche
- BP\_AusgleichsMassnahme
- BP\_EmissionskontingentLaerm
- BP\_EmissionskontingentLaermGebiet
- BP\_ZusatzkontingentLaerm
- BP\_Richtungssektor
- BP\_ZusatzkontingentLaermFlaeche
- BP\_RichtungssektorGrenze → GML-Geometrie
- BP\_StrassenbegrenzungsLinie → GML-Geometrie

**Problem-Klassen:**

- XP\_Rasterdarstellung
  - nicht vorhanden in Datenbank
- XP\_GenerAttribut
  - nicht vorhanden in Datenbank
- Attribut abweichungText von BP\_BaugebietsTeilFlaeche
  - nicht vorhanden in Datenbank
- Attribut xplan:position von BP\_RichtungssektorGrenze (exp\_riSekGre()) und BP\_StrassenbegrenzungsLinie (exp\_strbegrLin())
  - GML-Geometrie Linie noch nicht realisiert
  - XPExport.py Z. 1869 u. 2501