

# ArchiCAD2eLCA\_comps

***Effiziente Bauteilschichtanalyse für das Life-Cycle-Assessment***

**Geschrieben von:**

John Herbert Bergmann  
Matrikelnummer: 433792

**Prüfer:**

Univ.-Prof. Dr. Jakob Beetz  
Lehrgebiet für Computergestütztes Entwerfen (DC)  
Prof. Dr. Linda Hildebrand  
Juniorprofessur für Rezykliergerechtes Bauen

## **Eidesstattliche Versicherung**

Hiermit erkläre ich, John Herbert Bergmann, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

---

Aachen, June 28, 2024

## **Kontakt**

RWTH Aachen University  
Design computation / Computergestütztes Entwerfen  
Schinkelstraße 1, 52062 Aachen  
<http://dc.rwth-aachen.de>

John Herbert Bergmann  
Matrikelnummer: 433792  
Email: [john.bergmann@rwth-aachen.de](mailto:john.bergmann@rwth-aachen.de)

## Vorwort

An dieser Stelle möchte ich mich bei allen bedanken, die mich während meiner Bachelorthesis unterstützt haben.

Zunächst möchte ich Prof. Dr. Jakob Beetz meinen aufrichtigen Dank aussprechen. Er hat mich für Building Information Modeling (BIM) begeistert und stets meinen Horizont erweitert. Seine inspirierende Lehrweise und sein tiefgehendes Wissen haben mir nicht nur das notwendige theoretische Fundament vermittelt, sondern auch meine Leidenschaft für das Thema entfacht. Ohne seine fortwährende Unterstützung und seine wertvollen Anregungen wäre diese Arbeit nicht in der vorliegenden Form möglich gewesen.

Ein besonderer Dank gilt meinem Freund Aaron Neugebauer. Aaron war für mich nicht nur eine konstante Unterstützung über viele Jahre hinweg, sondern er hat mir auch die Grundlagen der Programmierung beigebracht. Seine Geduld und sein Engagement, mir komplizierte Konzepte verständlich zu machen, waren von unschätzbarem Wert. Darüber hinaus half er mir flexibel und zuverlässig bei der Betreuung meines Codes. Seine Hilfsbereitschaft und sein technisches Know-how haben wesentlich dazu beigetragen, dass diese Arbeit erfolgreich abgeschlossen werden konnte.

Des Weiteren möchte ich Thomas Stachelhaus meinen tiefen Dank aussprechen. Thomas war stets ein offenes Ohr für mich und bot mir kontinuierlich Unterstützung und Vertrauen am Lehrstuhl. Seine Bereitschaft, mir bei Problemen und Fragen zu helfen, sowie sein uningeschränktes Vertrauen in meine Fähigkeiten, haben mir während der gesamten Thesis wertvolle Sicherheit und Rückhalt gegeben.

Zum Schluss möchte ich meinen Eltern einen ganz besonderen Dank aussprechen. Sie haben mich über die Jahre während meiner gesamten Ausbildung und meines Studiums uneingeschränkt unterstützt. Ihre beständige Ermutigung und ihr Glaube an mich haben mir die Kraft und die Motivation gegeben, meine Ziele zu verfolgen und letztendlich zu erreichen. Ohne ihre Unterstützung wäre all dies nicht möglich gewesen. Ihnen verdanke ich nicht nur meine akademischen Erfolge, sondern auch die Möglichkeit, meine Träume zu verwirklichen.

Mit tiefem Dank und großer Wertschätzung,  
John Herbert Bergmann

## Abstract

Die Bachelorthesis *ArchiCAD2eLCA\_comps: Effiziente Bauteilschichtanalyse für das Life-Cycle-Assessment* von John Herbert Bergmann untersucht die Integration von Building Information Modeling (BIM) und Life-Cycle-Assessment (LCA) zur Optimierung der Ökobilanzierung von Gebäuden. Im Mittelpunkt der Arbeit steht die Entwicklung einer Methode zur Extraktion von Bauteilschichten aus Industry Foundation Classes (IFC)-Dateien, um diese für das LCA-Tool eLCA (Bauteileditor) nutzbar zu machen. Ziel ist es, den Prozess der Ökobilanzierung durch die Automatisierung der Datenerfassung und -verarbeitung zu vereinfachen und zu beschleunigen.

Die Thesis umfasst eine umfassende Literaturrecherche zum aktuellen Stand der Forschung in den Bereichen BIM und LCA, die Entwicklung und Implementierung eines Python-Skripts zur Identifikation und Konvertierung relevanter Bauteilschichten aus ArchiCAD-Modellen, sowie die Anwendung und Evaluation der entwickelten Methode in einem Fallbeispiel. Die Ergebnisse zeigen, dass die Methode den Aufwand für die Ökobilanzierung signifikant reduziert und somit einen wichtigen Beitrag zur Effizienzsteigerung und Nachhaltigkeit im Bauwesen leistet.

**Keywords:** Building Information Modeling, BIM, Life-Cycle-Assessment, LCA, Industry Foundation Classes, IFC, ArchiCAD, eLCA, Bauteilschichtanalyse, Nachhaltigkeit im Bauwesen, Ökobilanzierung, Bauinformatik

## Table of Contents

<b>Vorwort . . . . .</b>	<b>3</b>
<b>Abstract . . . . .</b>	<b>4</b>
<b>Inhaltsverzeichnis . . . . .</b>	<b>5</b>
<b>1 Einleitung . . . . .</b>	<b>6</b>
1.1 Motivation der Arbeit . . . . .	6
1.2 Forschungsansatz . . . . .	6
1.3 Verwandte Arbeiten . . . . .	7
1.4 Anvisierte Ergebnisse . . . . .	10
<b>2 Stand der Forschung . . . . .</b>	<b>10</b>
2.1 Ökobilanzierung . . . . .	11
2.2 Datenbasis für Ökobilanzierung . . . . .	11
2.3 BIM – Building Information Modeling . . . . .	12
2.4 Offenes Austauschformat – Industry Foundation Classes . . . . .	12
2.5 eLCA . . . . .	13
<b>3 Theoretische Grundlagen in eLCA . . . . .</b>	<b>13</b>
3.1 Projekterstellung . . . . .	13
3.2 Anlegen einer Bauteilvorlage . . . . .	14
3.3 Import von Bauteilvorlagen . . . . .	16
<b>4 Praktische Umsetzung und Anwendung der entwickelten Methode . . . . .</b>	<b>17</b>
4.1 Einrichtung und Konfiguration . . . . .	17
4.2 Analyse und Verarbeitung von IFC-Daten zur Erstellung von Bauteil-Objekten . . . . .	19
4.3 Erstellung von eLCA-XML-Dateien aus Bauteil-Objekten . . . . .	24
4.4 Anwendung als Command-Line-Tool . . . . .	25
<b>5 Fazit und Ausblick auf zukünftige Forschungsansätze . . . . .</b>	<b>25</b>
5.1 Analyse der Ergebnisse . . . . .	25
5.2 eLCA Baustoffe in ArchiCAD importieren . . . . .	26
5.3 Baustoffzuweisung mit Matching-Verfahren . . . . .	28
<b>Literaturverzeichnis . . . . .</b>	<b>30</b>
<b>Appendix . . . . .</b>	<b>32</b>

# 1 Einleitung

Die vorliegende Arbeit mit dem Titel *ArchiCAD2eLCA\_comps: Effiziente Bauteilschichtanalyse für das Life-Cycle-Assessment* befasst sich mit der Integration von Building Information Modeling (BIM) und Life-Cycle-Assessment (LCA) zur Verbesserung der Ökobilanzierung von Gebäuden. Diese Einleitung wird die Motivation der Arbeit, den Forschungsansatz, verwandte Arbeiten und die anvisierten Ergebnisse näher beleuchten.

## 1.1 Motivation der Arbeit

Die Bauindustrie steht vor der Herausforderung, nachhaltigere und umweltfreundlichere Baupraktiken zu entwickeln. Angesichts des steigenden Bewusstseins für den Klimawandel und die Notwendigkeit, Ressourcen effizienter zu nutzen, gewinnt das Konzept der Ökobilanzierung (LCA) an Bedeutung. LCA ermöglicht es, die Umweltauswirkungen eines Gebäudes über seinen gesamten Lebenszyklus hinweg zu bewerten – von der Rohstoffgewinnung über die Nutzung bis hin zur Entsorgung. Trotz der Vorteile, die LCA bietet, sind die derzeitigen Verfahren oft mit hohem Aufwand verbunden, da die Daten manuell erfasst und verarbeitet werden müssen. Dies führt zu Zeitverzögerungen und potenziellen Fehlern. Durch die Integration von BIM, einer digitalen Darstellung der physischen und funktionalen Eigenschaften eines Gebäudes, mit LCA soll dieser Prozess effizienter und genauer gestaltet werden. Diese Arbeit zielt darauf ab, eine Methode zu entwickeln, die es ermöglicht, Bauteilschichten aus IFC-Dateien (Industry Foundation Classes) zu extrahieren und für das LCA-Tool eLCA aufzubereiten. Dadurch soll der gesamte Prozess der Ökobilanzierung vereinfacht und beschleunigt werden.

## 1.2 Forschungsansatz

Der Forschungsansatz dieser Arbeit gliedert sich in mehrere Phasen. Zunächst wird eine umfassende Literaturrecherche durchgeführt, um den aktuellen Stand der Forschung im Bereich BIM und LCA zu erfassen. Dabei werden bestehende Methoden und Tools analysiert sowie die Herausforderungen und Potenziale der Integration von BIM und LCA untersucht. Auf dieser Grundlage wird ein Konzept für die Entwicklung eines Tools zur automatisierten Extraktion und Analyse von Bauteilschichten aus BIM-Modellen erarbeitet.

Im praktischen Teil der Arbeit wird dieses Tool implementiert. Dazu wird ein Python-Skript mit der IfcOpenShell-Bibliothek verwendet, um die Bauteilschichten aus den IFC-Dateien zu identifizieren und in ein kompatibles Format für das LCA-Tool eLCA zu überführen. Hierbei wird besonderer Wert auf die Genauigkeit und Effizienz des Extraktionsprozesses gelegt. Anschließend wird die entwickelte Methode in einem Fallbeispiel angewendet und evaluiert. Durch die praktische Umsetzung und Anwendung der Methode sollen die theoretischen Überlegungen und Konzepte

überprüft und validiert werden.

### 1.3 Verwandte Arbeiten

#### **Klassifizierung von Bauteilen auf Grundlage der DIN SPEC 91400 in ARCHICAD<sup>1</sup>**

<sup>1</sup>Göbels (2020)

Göbels hat einen Prototyp entwickeln, der Bauteile in ArchiCAD automatisch gemäß dem Klassifizierungssystem der DIN SPEC 91400 klassifiziert und dessen Merkmalsets verfügbar macht. Darüber hinaus sollte der Prototyp die Zuordnung der klassifizierten Bauteile zu DIN 276 Kostengruppen und IFC-Typen ermöglichen, um Baukostenberechnungen und Leistungsverzeichnisse direkt auswerten zu können.

Die erforderlichen Daten aus der DIN SPEC 91400 wurden über einen Suchprozess gefiltert und anschließend in ein ArchiCAD-kompatibles Format (XML-Datei) umgewandelt. Diese XML-Dateien wurden in ArchiCAD importiert und bestimmte Merkmale wurden in ArchiCAD-interne Eigenschaften übersetzt, um doppelte Angaben und Mehraufwand zu vermeiden.

Der entwickelte Prototyp ermöglicht die Klassifizierung von Bauteilen direkt in ArchiCAD und deren Zuordnung zu Kostengruppen und IFC-Typen. Dies reduzierte mögliche Fehlerquellen und optimierte die Baukostenberechnung und Erstellung von Leistungsverzeichnissen durch die direkte Integration von Ordnungssystemen und Datensätzen in die BIM-Software

Anne Göbels schlussfolgerte, dass die direkte Einbindung der DIN SPEC 91400 in ArchiCAD eine erhebliche Aufwandsminimierung darstellt und Fehlerquellen durch die Reduzierung von Austauschprozessen minimiert werden. Zukünftige Arbeiten könnten sich auf die Entwicklung von automatisierten Klassifizierungsprozessen konzentrieren, sowohl halb- als auch vollautomatisch, um die Effizienz weiter zu steigern und die Softwareunabhängigkeit zu erhöhen.

#### **Ökobilanzierung und BIM im Nachhaltigen Bauen<sup>2</sup>**

<sup>2</sup>Lambertz et al. (2019)

Das Projekt *Ökobilanzierung und BIM im Nachhaltigen Bauen* des Bundesinstituts für Bau-, Stadt- und Raumforschung (BBSR) zielt darauf ab, die Integration von Ökobilanzdaten in den Building Information Modeling (BIM) Prozess zu verbessern. Dies soll sicherstellen, dass ökologische Daten effizient und fehlerfrei zwischen den Beteiligten ausgetauscht werden können, um nachhaltiges Bauen zu fördern. Im Rahmen der Studie wurden mehrere methodische Schritte unternommen.

Zunächst wurde eine umfassende Analyse der ÖKOBAUDAT durchgeführt. Diese Datenbank enthält Umweltindikatoren und Rechenverfahren für Gebäudeökobilanzen und bildet die Grundlage für die Integration der Ökobilanzdaten in den BIM-Prozess. Im nächsten Schritt wurden verschiedene Anbindungsoptionen bewertet. Dazu gehörten die Nutzung

vorhandener *IfcPropertySets*, die Erweiterung des IFC-Datenmodells, das Referenzieren mit bestehenden Datenformaten aus ÖKOBAUDAT/eLCA, direkte API-Schnittstellen und die Integration von Ökobilanzdaten in BIM-Objekte.

Eine besondere Herausforderung bestand in der Definition des Informationsflusses und der Validierung der Daten. Hierzu wurden die Informationslieferungszeitpunkte und die notwendigen Datenmengen präzise festgelegt. Es kamen standardisierte Methoden wie das Information Delivery Manual (IDM) und die Model View Definition (MVD) zum Einsatz, um die technische Überprüfbarkeit der Daten sicherzustellen.

Die Ergebnisse der Studie zeigen, dass die Kombination der Optionen „Referenzieren mit bestehenden Datenformaten“ und „Direkte API-Schnittstelle“ besonders geeignet ist, um Ökobilanzdaten in den BIM-Prozess zu integrieren. Diese Ansätze ermöglichen eine praktikable und effiziente Lösung für den Datenaustausch. Durch die Anwendung der standardisierten Methoden konnten detaillierte Prozessdiagramme und technische Anleitungen erstellt werden, die den Informationsaustausch im BIM-Prozess optimieren.

Darüber hinaus wurden Mustertextblöcke und Modellierungshinweise für Auftraggeber-Informationsanforderungen (AIA) und BIM-Abwicklungspläne (BAP) entwickelt, um die BIM-Ökobilanzierung in Bundesbaumaßnahmen zu verankern. Diese Standards und Prozesse tragen dazu bei, die ökologische Qualität von Gebäuden schon in frühen Planungsphasen sicherzustellen.

Insgesamt zeigt die Arbeit, dass die entwickelten Methoden und Standards eine effiziente Integration von Ökobilanzdaten in den BIM-Prozess ermöglichen. Für die Zukunft wird empfohlen, diese Standards und Prozesse kontinuierlich weiterzuentwickeln und an neue technische und ökologische Anforderungen anzupassen, um nachhaltige Bauprojekte weiter zu fördern.

### **BIM-LCA Integration – A conceptual database alignment and conformity model<sup>3</sup>**

<sup>3</sup>Bangalu and Beetz (n.d.)

Die Arbeit von Apollos Yohanna Bangalu und Jakob Beetz konzentriert sich auf die Integration von Building Information Modeling und Life Cycle Assessment durch die Entwicklung eines konzeptuellen Datenbankabgleichs- und Konformitätsmodells. Ziel ist es, die unterschiedlichen Datenstrukturen und Anforderungen beider Disziplinen aufeinander abzustimmen, um eine nahtlose und effiziente Datennutzung zu ermöglichen. Im Kern dieser Arbeit steht die Identifizierung und Analyse relevanter Datenfelder und -formate in BIM und LCA. Anschließend wird ein Abgleichsprozess entwickelt, der die Konvertierung und Anpassung von Daten sicherstellt, sodass die in BIM gespeicherten Informationen den Anforderungen der LCA entsprechen und umgekehrt.

Ein zentrales Element des Modells ist die Nutzung der Industry Founda-

tion Classes als standardisiertes Austauschformat, um die Interoperabilität zwischen verschiedenen Softwarelösungen zu gewährleisten. Durch die Verwendung von IFC können BIM-Daten in einem strukturierten und offenen Format exportiert und für die LCA-Analyse genutzt werden. Dies erfordert jedoch die Erweiterung und Anpassung der IFC-Datenmodelle, um zusätzliche LCA-spezifische Informationen zu integrieren.

Die Arbeit beschreibt auch die Entwicklung von Validierungsmechanismen, um die Konformität der Daten mit den jeweiligen Anforderungen sicherzustellen. Dazu gehören Prüfregeln und Algorithmen, die die Konsistenz und Vollständigkeit der Daten überprüfen und mögliche Inkonsistenzen oder Fehler identifizieren. Ein definierter Workflow beschreibt die schrittweise Integration und Nutzung der Daten von BIM zu LCA, von der Datenextraktion und -transformation bis hin zur Analyse und Berichterstattung der LCA-Ergebnisse. Die Arbeit von Bangalu und Beetz beschränkt sich auf Beton-Baustoffe und vergleicht Materialien aus ArchiCAD und Revit.

Durch dieses strukturierte und systematische Vorgehen bietet die Arbeit von Beetz und Bangalu eine solide Grundlage für die Integration von BIM und LCA. Dies führt zu einer verbesserten Effizienz und Genauigkeit bei der Bewertung der Umweltleistung von Bauprojekten, indem es die umfassende Nutzung von BIM-Daten für die ökologische Bewertung ermöglicht.

### **Gebäude als Materialressource – OpenBIM in der Ökobilanzierung<sup>4</sup>**

<sup>4</sup>Schneider (2021)

Jil Schneider hat sich zum Ziel gesetzt, die Effizienz und Nachhaltigkeit im Bauwesen durch die Integration von BIM und LCA zu verbessern. Der Fokus liegt auf der Optimierung des Bauprozesses hinsichtlich Ressourcenverbrauch und Umweltbelastung durch die Verwendung digitaler Modelle und analytischer Methoden.

Um dieses Ziel zu erreichen, hat Schneider eine Methode entwickelt, die den Import von IFC-Dateien in das LCA-Tool eLCA ermöglicht. Die Methodik umfasst zunächst eine detaillierte Analyse und Kategorisierung von Bauteilen, basierend auf IFC-Daten. Anschließend folgt eine Filterung und Klassifizierung der Bauteile nach DIN 276 und Zuweisung zu Kostengruppen. Abschließend wird eine Lebenszyklusanalyse auf Basis der aufbereiteten Daten durchgeführt, um die ökologischen Auswirkungen der Bauteile zu bewerten.

Schneider stellte fest, dass die Integration von BIM und LCA signifikante Vorteile bietet. Durch die Automatisierung der Datenerfassung und -aufbereitung wird der Aufwand für die Erstellung von LCA reduziert. Die Verwendung von genauen digitalen Modellen führt zu präziseren Analysen und Bewertungen und verbesserte Entscheidungsfindung im Bauprozess basierend auf quantifizierten ökologischen Auswirkungen.

In ihrem Resultat beschreibt Schneider, dass für den Import von Bauteilvorlagen in einem automatisierten Prozess aus dem IFC-Modell erst eine

funktionierende Schnittstelle zu den Ökobilanzdaten der ÖKOBAUDAT in die BIM-Software geschaffen werden müsste.

Die Arbeit zeigt, dass eine engere Verzahnung von BIM und LCA ein vielversprechender Ansatz ist, um die Nachhaltigkeit im Bauwesen zu fördern. Schneider empfiehlt, diese Methoden weiterzuentwickeln und in die Praxis zu integrieren, um langfristig eine nachhaltigere Bauwirtschaft zu erreichen. Künftige Arbeiten könnten sich darauf konzentrieren, die Benutzerfreundlichkeit der Tools zu verbessern und die Integration in bestehende Bauprozesse zu erleichtern.

**“Um die Bauteilvorlagen in einem automatisierten Prozess aus dem IFC-Modell zu importieren, muss erst eine funktionierende Schnittstelle zur Integration der Ökobilanzdaten der ÖKOBAUDAT in die BIM-Software geschaffen werden.”**

— Schneider, 2021

## 1.4 Anvisierte Ergebnisse

Ziel dieser Arbeit ist es, eine Methode zur automatisierten Extraktion von Bauteilschichten aus BIM-Modellen zu entwickeln und deren Effizienz und Genauigkeit zu demonstrieren. Es wird erwartet, dass die entwickelte Methode den Aufwand für die Erstellung von Ökobilanzen signifikant reduziert, indem sie den manuellen Prozess der Datenerfassung und -verarbeitung automatisiert. Dadurch sollen die Genauigkeit und Konsistenz der Daten verbessert und potenzielle Fehlerquellen minimiert werden. Die Ergebnisse dieser Arbeit sollen zeigen, dass die Integration von BIM und LCA durch die entwickelte Methode einen wichtigen Beitrag zur Effizienzsteigerung und Nachhaltigkeit im Bauwesen leisten kann.

Darüber hinaus sollen die Ergebnisse dieser Arbeit als Grundlage für weitere Forschungen und Entwicklungen in diesem Bereich dienen. Zukünftige Arbeiten könnten sich darauf konzentrieren, die Methode weiter zu verfeinern und anzupassen, um ihre Anwendbarkeit und Effizienz noch weiter zu steigern. Insbesondere die Entwicklung von Standards und Schnittstellen für den Datenaustausch zwischen BIM- und LCA-Tools könnte ein vielversprechender Ansatz sein, um die Integration dieser Technologien weiter zu fördern.

Insgesamt zielt diese Arbeit darauf ab, den Prozess der Ökobilanzierung von Gebäuden durch die Kombination von BIM und LCA zu revolutionieren und dadurch einen Beitrag zu einer nachhaltigeren Bauindustrie zu leisten.

## 2 Stand der Forschung

Die fortschreitende Digitalisierung im Bauwesen bringt zahlreiche Vorteile mit sich, darunter eine verbesserte Effizienz und Genauigkeit in der Planung und Ausführung von Bauprojekten. Ein zentrales Thema dabei ist die Ökobilanzierung von Gebäuden, die eine ganzheitliche Bewertung der Umweltauswirkungen über den gesamten Lebenszyklus eines Bauwerks ermöglicht. Die folgenden Unterpunkte bieten einen Überblick über den aktuellen Stand der Forschung in diesem Bereich.

## 2.1 Ökobilanzierung

Ökobilanzierung, auch bekannt als Lebenszyklusanalyse (LCA), ist eine Methode zur Bewertung der Umweltwirkungen eines Produkts oder einer Dienstleistung über dessen gesamten Lebenszyklus hinweg. Im Kontext der Bauindustrie umfasst die Ökobilanzierung die Bewertung der Umweltbelastungen, die mit der Gewinnung von Rohstoffen, der Herstellung von Baumaterialien, dem Bauprozess, der Nutzung und dem Abriss eines Gebäudes verbunden sind. Ziel ist es, alle relevanten Umweltaspekte zu identifizieren und zu quantifizieren, um eine fundierte Basis für umweltgerechte Entscheidungen zu schaffen<sup>5</sup>.

<sup>5</sup>Baumann and Tillman (2004)

Gebäude haben einen erheblichen Einfluss auf die Umwelt, sei es durch den Ressourcenverbrauch, die Energieeffizienz oder die Emission von Treibhausgasen. Eine umfassende Ökobilanz hilft dabei, diese Einflüsse zu quantifizieren und Maßnahmen zur Reduktion zu identifizieren<sup>6</sup>.

<sup>6</sup>“DIN EN ISO 14040” (2021)

Durch die Analyse und Optimierung der Umweltwirkungen von Baumaterialien und Bauprozessen kann die Bauindustrie nachhaltigere Methoden entwickeln und umsetzen, was zu einer Verringerung des ökologischen Fußabdrucks führt<sup>7</sup>.

<sup>7</sup>Kibert (2016)

Viele Länder und Regionen haben gesetzliche Vorgaben oder bieten Anreize für nachhaltiges Bauen. Zudem sind Zertifizierungssysteme wie LEED, BREEAM oder DGNB darauf angewiesen, dass genaue Ökobilanzdaten vorliegen, um Gebäude bewerten zu können<sup>8</sup>.

<sup>8</sup>Cole (2005)

Eine fundierte Ökobilanzierung kann auch wirtschaftliche Vorteile bieten, indem sie hilft, effizientere und weniger materialintensive Bauprozesse zu entwickeln, was langfristig zu Kosteneinsparungen führen kann<sup>9</sup>.

<sup>9</sup>Dodoo et al. (2014)

## 2.2 Datenbasis für Ökobilanzierung

Für eine fundierte Ökobilanzierung müssen verschiedene Daten vorliegen. In Deutschland ist die ÖKOBAUDAT eine zentrale Datenbank, die zu allen genannten Punkten Ökobilanzdaten für Bauprodukte bereitstellt<sup>10</sup>.

<sup>10</sup>BMUB (2017)

**Rohstoff- und Materialdaten:** Informationen über die Art und Menge der verwendeten Rohstoffe und Baumaterialien sowie deren Produktionsprozesse. Dies umfasst Daten zur Energie- und Wasserverwendung, Emissionen und Abfällen bei der Materialherstellung<sup>11</sup>.

<sup>11</sup>BMUB (2017)

**Bauprozessdaten:** Daten zum Energie- und Ressourcenverbrauch während des Baus, einschließlich der Transportwege und -mittel, die zur Baustelle führen, sowie die Maschinen und Geräte, die während des Bauprozesses verwendet werden<sup>12</sup>.

<sup>12</sup>BMUB (2017)

**Nutzungsphase:** Informationen über den Energieverbrauch, die Wartung und Reparaturmaßnahmen sowie die Renovierungszyklen während der gesamten Lebensdauer des Gebäudes. Diese Daten helfen, die

Umweltwirkungen während der Betriebsphase des Gebäudes zu bewerten<sup>13</sup>.

<sup>13</sup>Bribian et al. (2011)

**End-of-Life-Daten:** Daten zur Entsorgung oder Wiederverwertung von Materialien nach dem Abriss des Gebäudes. Dies umfasst die Methoden der Abfallbehandlung und die potenziellen Umweltauswirkungen der Entsorgungsprozesse<sup>14</sup>.

<sup>14</sup>BMUB (2017)

**Umweltwirkungen:** Daten zu den Umweltwirkungen der verschiedenen Prozesse, wie z.B. Treibhausgasemissionen, Versauerungspotenzial, Eutrophierungspotenzial, Ozonabbaupotenzial, etc<sup>15</sup>.

<sup>15</sup>Wernet et al. (2016)

## 2.3 BIM – Building Information Modeling

Building Information Modeling (BIM) ist eine kooperative Arbeitsmethodik, die die Nutzung digitaler Modelle eines Bauwerks über dessen gesamten Lebenszyklus hinweg umfasst. Diese Modelle beinhalten die dreidimensionale Geometrie, nicht-physische Objekte wie Räume und Zonen sowie eine hierarchische Projektstruktur<sup>16</sup>. BIM ermöglicht es, verschiedene Aspekte eines Bauwerks, wie Architektur, Struktur, Mechanik, Elektrik und Sanitär, in einem integrierten Modell zu vereinen. Eine der Hauptstärken von BIM liegt in der durchgängigen Nutzung und verlustarmen Weitergabe eines digitalen Bauwerksmodells, was die Wiedereingabe von Informationen reduziert, die Kommunikation verbessert und die Zusammenarbeit zwischen allen Beteiligten effizienter gestaltet<sup>17</sup>. BIM unterstützt zudem die Simulation und Analyse von Bauabläufen, Energieeffizienz, Kosten und Zeitplänen, was zu besseren Planungs- und Entscheidungsprozessen führt. Das BIM-Modell verwendet IFC, um Daten zwischen verschiedenen Softwareplattformen auszutauschen, wodurch eine nahtlose Zusammenarbeit und Datentransparenz gewährleistet wird<sup>18</sup>.

<sup>16</sup>Borrmann et al. (2021)

<sup>17</sup>Borrmann et al. (2021)

<sup>18</sup>Borrmann et al. (2021)

## 2.4 Offenes Austauschformat – Industry Foundation Classes

Industry Foundation Classes (IFC) sind ein offenes, standardisiertes Datenformat für den digitalen Austausch von Bauwerksinformationen. IFC ermöglicht die Interoperabilität zwischen verschiedenen Softwareanwendungen und dient als gemeinsame Sprache zur Beschreibung von Bauwerksdaten. Die Stärken von IFC liegen in der Möglichkeit, Informationen unabhängig von spezifischen Softwarelösungen auszutauschen, was die Zusammenarbeit und Koordination zwischen unterschiedlichen Disziplinen und Werkzeugen erleichtert und die Datenkonsistenz über verschiedene Lebenszyklusphasen unterstützt<sup>19</sup>. Die internationale Relevanz von IFC wird durch seine Standardisierung und globale Akzeptanz unterstrichen, wodurch es den weltweiten Austausch und die Zusammenarbeit in der Bauindustrie fördert. IFC fungiert dabei als Rückgrat für den Datenaustausch innerhalb von BIM-Prozessen, da es sicherstellt, dass die umfangreichen Informationen, die in einem BIM-Modell enthalten sind, unabhängig von den verwendeten Softwareplattformen einheitlich und korrekt weitergegeben werden können<sup>20</sup>.

<sup>19</sup>Borrmann et al. (2021)

<sup>20</sup>Borrmann et al. (2021)

## 2.5 eLCA

eLCA ist ein spezialisiertes Online-Tool zur Durchführung von Ökobilanzierungen im Bauwesen, das von der Deutschen Gesellschaft für Nachhaltiges Bauen (DGNB) und dem Bundesministerium für Umwelt, Naturschutz, Bau und Reaktorsicherheit (BMUB) entwickelt wurde. Es ermöglicht die umfassende Bewertung der ökologischen Auswirkungen von Bauprojekten über deren gesamten Lebenszyklus hinweg. eLCA unterscheidet sich vor allem in drei Aspekten von der Konkurrenz.

Der erste Aspekt ist die direkte Integration der ÖKOBAUDAT, einer umfangreichen Datenbank, die spezifische Ökobilanzdaten für Bauprodukte bereitstellt. Diese Datenbank wird vom BMUB verwaltet und bietet aktuelle, verlässliche und umfassende ökologische Daten zu Baumaterialien und Bauprozessen in Deutschland<sup>21</sup>. Da die ÖKOBAUDAT direkt in eLCA eingebunden ist, können Benutzer schnell und einfach auf diese umfangreiche Datensammlung zugreifen, was die Genauigkeit und Effizienz der Ökobilanzierung erheblich verbessert<sup>22</sup>.

<sup>21</sup>BMUB (2017)

<sup>22</sup>BMUB (2017)

Der zweite Aspekt ist die Spezialisierung auf das Bauwesen. Während viele andere LCA-Werkzeuge, wie SimaPro, GaBi oder OpenLCA, für eine Vielzahl von Industrien und Anwendungen entwickelt wurden, ist eLCA speziell auf die Bedürfnisse der Bauindustrie zugeschnitten. Dies ermöglicht eine gezieltere und detailliertere Analyse der spezifischen Umweltwirkungen von Bauprojekten<sup>23</sup>. eLCA wurde so konzipiert, dass es besonders benutzerfreundlich für Fachleute im Bauwesen ist, die möglicherweise nicht die tiefgehende Expertise in der Lebenszyklusanalyse haben, die für die Verwendung allgemeinerer LCA-Werkzeuge erforderlich ist<sup>24</sup>.

<sup>23</sup>ILCD (2010)

<sup>24</sup>DGNB (2020)

Der letzte Aspekt ist die Konformität mit deutschen Normen und Zertifizierungssystemen. eLCA unterstützt die Einhaltung spezifischer deutscher und europäischer Normen und Vorschriften im Bereich der Nachhaltigkeit und Ökobilanzierung von Gebäuden. Dies umfasst Richtlinien wie die DIN EN 15978, die die Bewertung der Umweltleistung von Gebäuden festlegt<sup>25</sup>.

<sup>25</sup>„DIN EN 15978“ (2011)

## 3 Theoretische Grundlagen in eLCA

Um die praktischen Methoden und deren Optimierung besser zu verstehen, müssen zunächst die theoretischen Grundlagen für die Erstellung eines Projekts, das Anlegen von Bauteilvorlagen und deren Import in eLCA erklärt werden.

### 3.1 Projekterstellung

Bei der Erstellung eines neuen Projektes stehen fünf Optionen zur Auswahl<sup>26</sup>. Für die Arbeitsweise mit einer BIM-Software ist vor allem die Option IFC-Import besonders relevant.

<sup>26</sup>Figure 1

[Neues Projekt +](#) [Projekt importieren +](#) [EnEV Projekt importieren +](#) [CSV/XLS Import +](#) [IFC Import +](#)

**Figure 1: Browseransicht: Erstellung Optionen zur Erstellung eines neuen Projekts in eLCA.**

Mit dieser Methode werden dem Benutzer alle IFC-Elemente des importierten IFC untereinander ausgegeben. Diesen Elementen müssen nun eine DIN 276 Kostengruppe, eine Menge und eine Bauteilvorlage zugewiesen werden<sup>27</sup>. Jedes Element, das nicht mit diesen drei Parametern ausgezeichnet wurde, wird aus dem Projekt entfernt<sup>28</sup>. Es ist also wichtig, dass diese Informationen bekannt sind, bevor der Import durchgeführt wird.

DIN 276	Name	Menge	Bauteilvorlage
330 - Außenwände	Wand-003	0 Stk	<input type="button" value="auswählen"/>
331 - Tragende Außenwän...	IfcWall		

GUID: 2gz\_eheCZUHPQup1i7XfAK

**Figure 2: Browseransicht: Importiertes IFC-Element in eLCA.**

### 3.2 Anlegen einer Bauteilvorlage

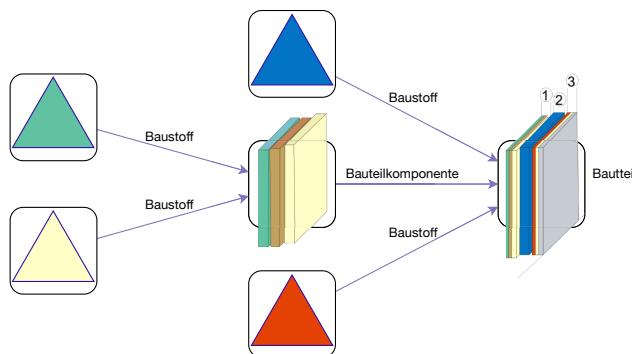
Materialien werden in eLCA alle aus der ÖKOBAUDAT gezogen und können dann verwendet werden, um daraus sogenannte Bauteilkomponenten zu erstellen oder einem Bauteil zuzuweisen. Bauteilkomponenten setzen sich also aus Materialschichten zusammen und werden auf der dritten Ebene der DIN 276 verortet<sup>29</sup>. Bauteile sind eine Komposition aus Materialien und/oder Bauteilkomponenten und werden auf der zweiten Ebene der DIN 276 verortet<sup>30</sup>. Beispielsweise ist ein Sandwichpanel, bestehend aus Aluminium, Dämmung und Folie, einer Fassade ein Element, das auf der Baustelle einer Fassade angefügt wird, doch nicht die Fassade selbst. Somit ist das Sandwichpanel eine Bauteilkomponente und alle Schichten der Fassade zusammen ein Bauteil. Die Bauteilkompositionen eines Bauteils können unterschiedliche Kostengruppen haben.

<sup>27</sup> Figure 2

<sup>28</sup> Schneider (2021)

<sup>29</sup> Rössig (2014)

<sup>30</sup> Rössig (2014)



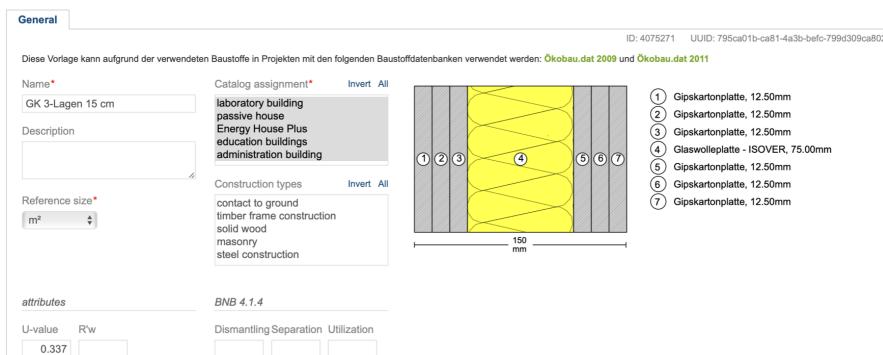
**Figure 3: Bauteilaufbau in eLCA.**

Bei der Erstellung einer Bauteilkomponente oder eines Bauteils muss

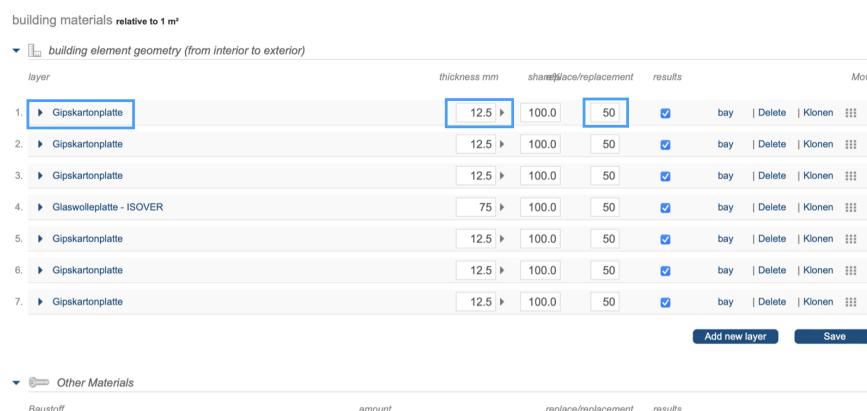
jedem Element zunächst eine DIN 276 Kostengruppe, ein Name, eine Bezugsgröße und eine Katalogzuordnung zugewiesen werden<sup>31</sup>. Die Bezugsgröße ist abhängig von der Berechnungseinheit für die Masse eines Elements. Zum Beispiel hat eine Wand eine Bezugsgröße von  $m^2$ , da die Masse abhängig ist von der Höhe und Länge der Wand. Die Dicke ergibt sich aus der Konfiguration der Materialschichten. Die Katalogzuordnung weist dem Element eine Gebäudekategorie zu, um nach diesen filtern zu können. Ein Element kann mehrere dieser Zuordnungen haben.

<sup>31</sup>Figure 4

Die Beschreibung, die Bauweise sowie die Attribute des Elements sind optionale Parameter und müssen bei der Erstellung nicht eingetragen werden.

**Figure 4: Browseransicht: Anlegen einer Bauteilvorlage in eLCA.**

Bei der Anlegung der Schichten können Materialien oder Bauteilkomponenten als neue Schichten hinzugefügt werden<sup>32</sup>. Dabei sind vor allem die Dicke der einzelnen Schichten und deren Nutzungsdauer relevant. Die Nutzungsdauer einer Schicht wird durch die Zeit in Jahren definiert, die eine Schicht haben darf, bis diese ausgetauscht werden muss<sup>33</sup>.

<sup>32</sup>Figure 5<sup>33</sup>eLCA Online Handbuch (2024)**Figure 5: Browseransicht: Anlegen von Schichten in einer Bauteilvorlage in eLCA.**

### 3.3 Import von Bauteilvorlagen

Bauteilvorlagen können als sogenannte XML-Dateien exportiert oder importiert werden.

Eine XML-Datei (Extensible Markup Language) ist ein textbasiertes Datenformat, das entwickelt wurde, um Daten strukturiert und lesbar zu speichern und zu transportieren. Sie verwendet Tags, die ähnlich wie HTML-Tags sind, um die Hierarchie und den Inhalt der Daten zu definieren. Eine typische XML-Datei beginnt mit einer XML-Deklaration und enthält ein *Root-Element*, das andere verschachtelte Elemente und Attribute umschließt<sup>34</sup>.

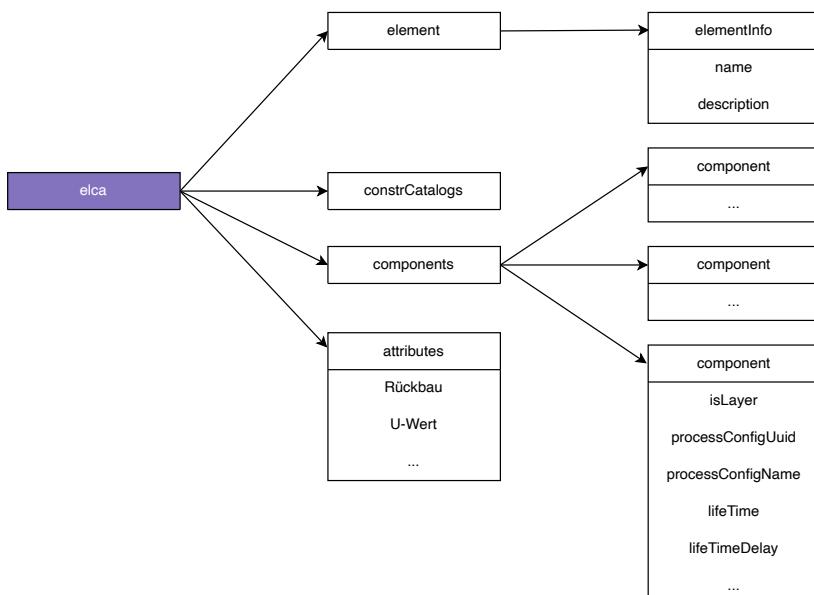
<sup>34</sup>Bray et al. (2008)

In einer exportierten Bauteilvorlage im XML-Format sind alle Informationen, die beim Anlegen gesetzt worden sind, in einem Baum gespeichert. Ein XML-Baum ist eine hierarchische Struktur, die die Anordnung der Elemente in einer XML-Datei darstellt, wobei jedes Element als Knoten in einem Baum fungiert, der ein übergeordnetes Element (Elternteil) oder untergeordnete Elemente (Kinder) haben kann. Diese Struktur ermöglicht eine organisierte und logische Darstellung der Daten<sup>35</sup>.

<sup>35</sup>Bray et al. (2008)

Der Aufbau einer XML-Bauteilvorlage beginnt mit einem *elca-Element*, das in seiner Struktur alle Informationen über die Eigenschaften und den Aufbau des Bauteils speichert<sup>36</sup>.

<sup>36</sup>Figure 6



**Figure 6: Aufbau der XML-Root einer exportierten Bauteilvorlage als Baum.**

## 4 Praktische Umsetzung und Anwendung der entwickelten Methode

Für die praktische Umsetzung der Ziele wurde ein benutzerdefiniertes *Property Set* für die BIM-Software ArchiCAD<sup>37</sup> und ein Python-Skript entwickelt.

Zuerst werden die ÖKOBAUDAT-Informationen in ArchiCAD-Baustoffen festgelegt. Dann werden aus dem exportierten IFC die Schichten mit Materialien analysiert und gespeichert. Für jedes einzigartige Bauteil im IFC wird eine XML-Datei erstellt. Diese XML-Datei kann als Bauteilvorlage in eLCA importiert werden.

<sup>37</sup> ArchiCAD ist eine BIM-Software, die von dem ungarischen Unternehmen Graphisoft entwickelt wurde. Die Software ist für Architekten und Bauingenieure, die 2D- und 3D-Planung, Modellierung und Dokumentation von Bauprojekten ermöglicht.

### 4.1 Einrichtung und Konfiguration

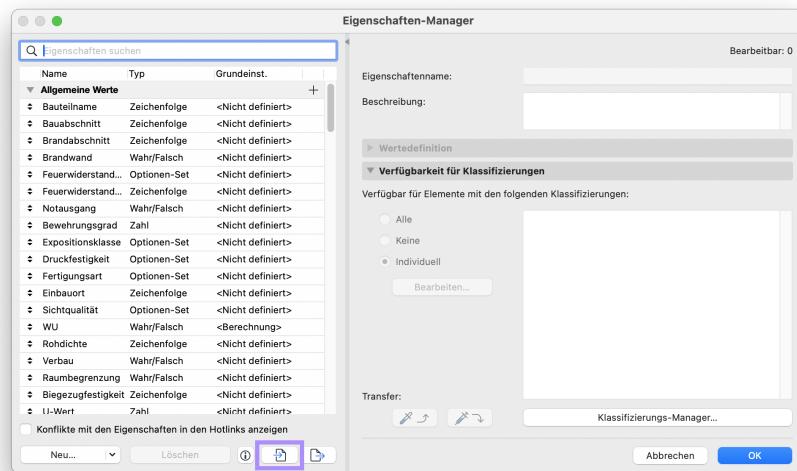
Einträge in der ÖKOBAUDAT erhalten einen Universally Unique Identifier (UUID). Eine UUID ist eine einzigartige, 36-stellige Zeichenfolge aus Kleinbuchstaben, Zahlen und Trennzeichen. Diese wird später zusammen mit der Nutzungsdauer in die XML-Datei der Bauteilvorlage geschrieben, um sie automatisch in eLCA mit den ÖKOBAUDAT-Informationen abzuleichen.

In ArchiCAD können Bauteilen eine oder mehrere Klassifizierungen zugewiesen werden. Diese Klassifizierungen bieten spezielle Parameter und Funktionen, die von anderen Anwendungen genutzt werden können, um Projektdaten zu verwalten, Modelle zu analysieren, Elemente zu suchen, Listen zu erstellen und Daten zu prüfen. Die Klassifizierung beeinflusst auch den IFC-Export, da jedem Element im IFC-Modell ein IFC-Typ gemäß seiner Klassifizierung zugeordnet wird<sup>38</sup>.

<sup>38</sup> ARCHICAD Klassifizierungen und Kategorien in IFC (2024)

Eigenschaften in ArchiCAD können an Klassifikationen von Baustoffen angehängt werden. Eigenschaften wie Dichte und Wärmekapazität von Baustoffen werden als *Property Set* an das Material angehängt. Da für die Attribute UUID und Nutzungsdauer eines Materials kein standardmäßiges *Property Set* in ArchiCAD vorhanden ist, kann dieses als benutzerdefiniertes *Property Set* importiert werden<sup>39</sup>. Für diesen Arbeitsschritt wird der Eigenschaften-Manager verwendet.

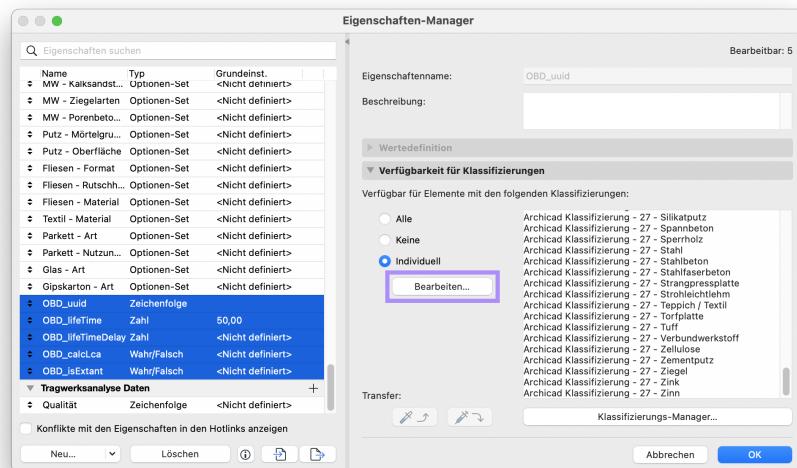
<sup>39</sup> Figure 7



**Figure 7: Screenshot ArchiCAD: Import Eigenschaften mit Eigenschaften-Manager**

Neu erstellte oder importierte Eigenschaften sind zunächst keiner Klassifizierung zugeordnet. Daher werden diese Eigenschaften in keinem Baustoff angezeigt. Im Klassifizierungs-Manager kann die Verfügbarkeit von Eigenschaften einer Klassifizierung bearbeitet werden<sup>40</sup>.

40 Figure 8



**Figure 8: Screenshot ArchiCAD: Verfügbarkeit von Eigenschaften in Klassifizierung bearbeiten.**

Die für den Export relevanten Parameter können nun in den ArchiCAD-Baustoffen gesetzt werden. Bei einem IFC-Export werden diese Eigenschaften als *Custom Property Set* im IFC gespeichert.

Werden die in den Projektdateien beigelegten Eigenschaften für den Import verwendet, muss dieser Schritt nicht beachtet werden, da die

Zuweisung der Eigenschaften an deren Klassifikationen bei dieser Datei in der deutschen Version von ArchiCAD automatisch passiert<sup>41</sup>.

Eine weitere wichtige Eigenschaft beim Export ist die IFC-Eigenschaft *ExtentToStructure* in der Klassifikation eines ArchiCAD-Elements. Diese Eigenschaft gibt an, ob sich das Objekt auf die darüber liegende Struktur erstreckt (TRUE) oder nicht (FALSE)<sup>42</sup> und wird benötigt, um die DIN 276 Kostengruppe des Elements zu bestimmen<sup>43</sup>.

Die IFC-Eigenschaften werden im Reiter "Klassifizierung und Eigenschaften" in den Einstellungen eines Bauteils geändert<sup>44</sup>. Dort befinden sich alle *Property Sets*, einer Klassifizierung, die beim IFC-Export des Bauteils berücksichtigt werden. Im *Property Set Pset\_WallCommon* kann *ExtentToStructure* aktiviert werden<sup>45</sup>.

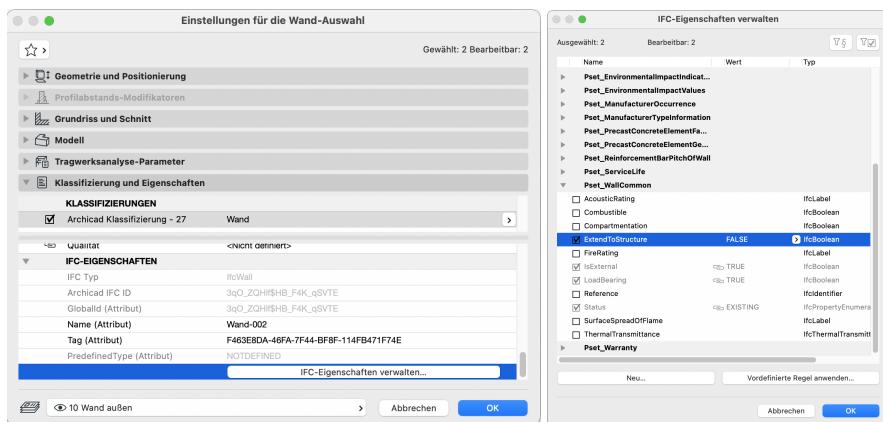
<sup>41</sup>Klassifizierungen und Eigenschaften sind in ArchiCAD je nach Sprach- und Ländereinstellungen anders.

<sup>42</sup>BuildingSMART (2024)

<sup>43</sup>Schneider (2021)

<sup>44</sup>Figure 9a

<sup>45</sup>Figure 9b



(a) Screenshot ArchiCAD: Klassifizierung und Eigenschaften in Bauteileinstellungen.

(b) Screenshot ArchiCAD: Verwaltung IFC-Eigenschaften.

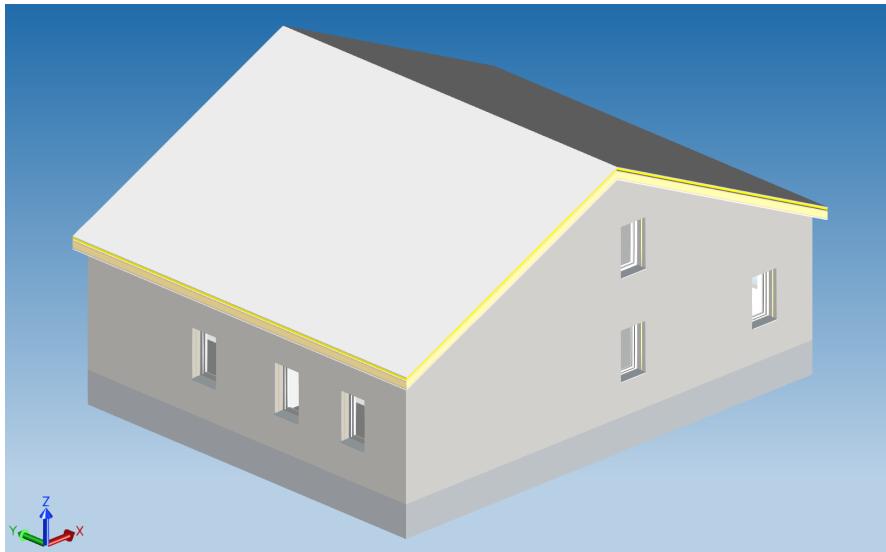
Figure 9: Screenshot ArchiCAD: Ifc-Eigenschaft ExtantToStructure für Pset\_WallCommon aktivieren.

## 4.2 Analyse und Verarbeitung von IFC-Daten zur Erstellung von Bauteil-Objekten

Beziehungen zwischen den verschiedenen Entitäten sind ein zentraler Bestandteil der IFC-Struktur. Diese Beziehungen definieren, wie die verschiedenen Komponenten des Modells miteinander verbunden sind<sup>46</sup>. Anhand eines Beispielgebäudes werden diese Beziehung im folgenden erläutert<sup>47</sup>.

<sup>46</sup>BuildingSMART (2024)

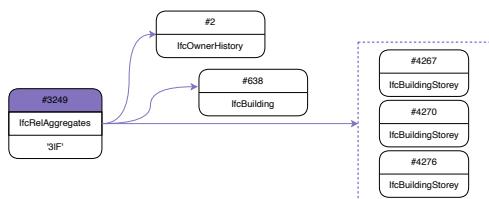
<sup>47</sup>Figure 10



**Figure 10:** Screenshot aus Open IFC Viewer: Zeigt Beispielgebäude.

Beispielsweise verbindet eine *IfcRelAggregates*-Entität ein Gebäude mit seinen Geschossen:

```
1 #3249 = IFCRELAGGREGATES('3lF', #2, $, $, #638, ((#4276, #426, #4270)
 );
```



**Figure 11:** Beziehungen von einem IfcRelAggregates als Graph.

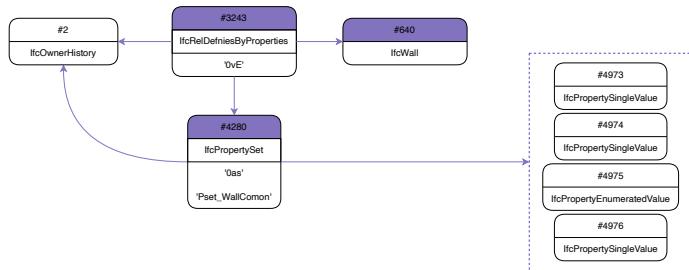
Die *IfcRelDefinesByProperties*-Entität stellt eine Beziehung zwischen einem IFC-Element (z.B. *IfcWall*) und einem Property Set (z.B. *Pset\_WallCommon*) her. Diese Beziehung wird durch eindeutige Identifikatoren (IDs) in der IFC-Datei festgelegt<sup>48</sup>. Ein Beispiel für eine solche Verknüpfung in der IFC-Syntax könnte wie folgt aussehen:

<sup>48</sup> BuildingSMART (2024)

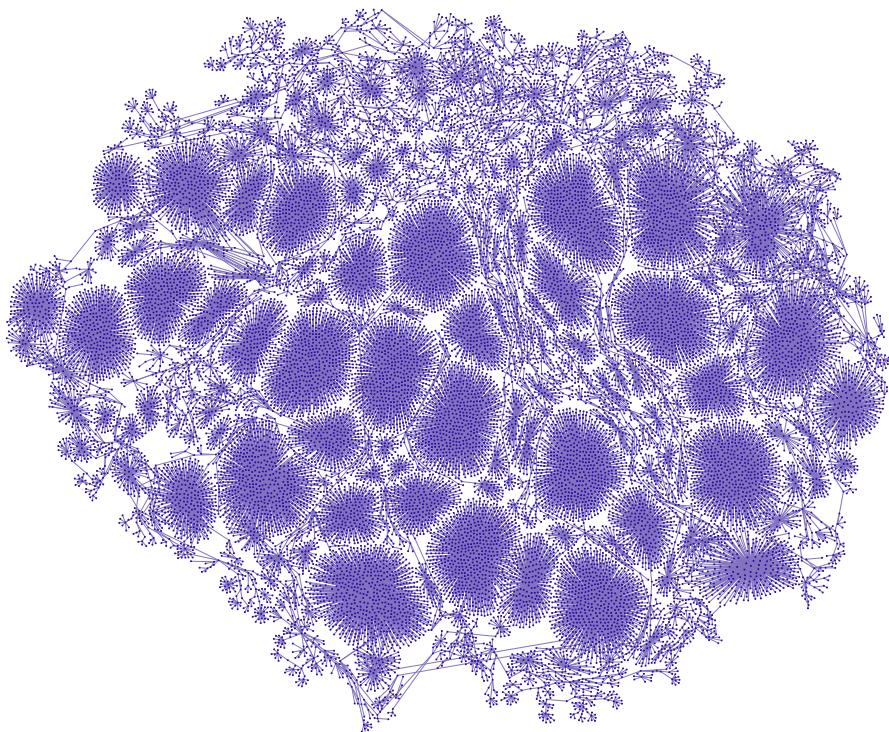
```
1 #4280 = IFCPROPERTYSET('0as', #2, 'Pset_WallCommon', $, (#4973,
 #4974, #4975, #4976));
2 #3243 = IFCRELDEFINESBYPROPERTIES('0vE', #2, $, $, (#640), #4280);
```

In diesem Beispiel ist das Property Set *Pset\_WallCommon* (mit der ID #4280) mit dem IFC-Element *IfcWall* (mit der ID #640) über die Entität *IfcRelDefinesByProperties* (mit der ID #3243) verknüpft<sup>49</sup>.

<sup>49</sup> Figure 12



**Figure 12: Beziehungen von einem IfcRelDefinesByProperties zu einem Pset\_WallCommon und einer IfcWall als Graph.**



**Figure 13: IFC Datei gerendert als Graph.**

Figure 13 zeigt den Graphen der IFC-Datei aus Figure 10. Jeder Knotenpunkt ist ein Eintrag in der IFC. Jede Linie ist eine Referenzierung eines Objektes auf ein anderes.

Um eine IFC-Datei auszulesen ist es demnach wichtig zu wissen, welche Informationen gesucht sind und wie diese abgespeichert werden. Als Methode diese Beziehungen zu analysieren wird IfcOpenShell verwendet. IfcOpenShell ist eine Python-Bibliothek, die es ermöglicht, Daten aus dem IFC-Format zu lesen, zu bearbeiten und zu visualisieren. Sie wird verwendet, um den Zugriff auf und die Manipulation von IFC-Dateien zu erleichtern, wodurch die Integration und Analyse von Bauwerksdaten in verschiedenen Softwareanwendungen verbessert wird<sup>50</sup>.

<sup>50</sup>IfcOpenShell (2024)

Gesuchte Eigenschaften eines Elements:

- Universally Unique Identifier (UUID)

- Name der Komposition
- Beschreibung der Kompositon
- Kostengruppe nach DIN 276
- Referenzeinheit

Gesuchte Eigenschaften einer Schicht eines Elements:

- Universally Unique Identifier (UUID) aus der
- Name des Materials
- Position des Materials in der Reihenfolge der Schichten
- Dicke [*m*]
- Nutzungsdauer [*a*]
- Verzögerung bis der Zeitraum der Nutzungsdauer startet [*a*]

In dem Codebeispiel dieser Arbeit werden derzeit ausschließlich Bauteilvorlagen aus *IfcWalls* analysiert.

Die UUID einer Bauteilvorlage kann problemlos zufällig generiert werden, da bei jedem Durchlauf ein neues Bauteil generiert wird und somit zu diesem noch keine UUID vorhanden ist. Um die Zuordnung zu einer DIN 276 Kostengruppe vorzunehmen, greift diese Arbeit auf einen Codeabschnitt von Jil Schneider zurück. In diesem Abschnitt wird das *Pset\_WallCommon* einer *IfcWall* ausgelesen, um zu entscheiden, zu welcher Kostengruppe die *IfcWall* gehört<sup>51</sup>. Beispielsweise ist eine außenliegende, tragende Wand, deren Struktur nicht über das Geschoss hinausragt Kostengruppe 331 Tragende Außenwände<sup>52</sup>.

<sup>51</sup>Schneider (2021)

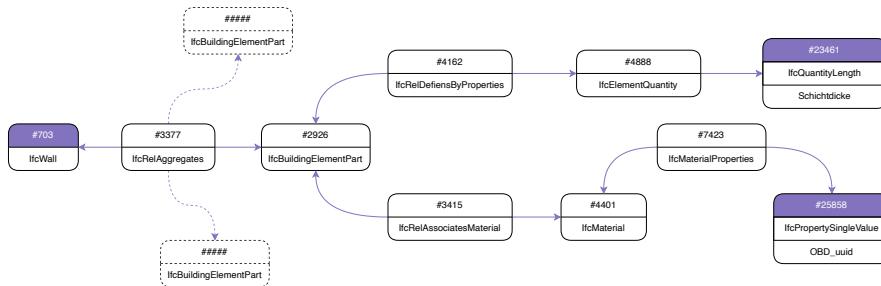
<sup>52</sup>“DIN 276” (2018)

Die Referenzgröße für eine *IfcWall* ist standardmäßig der Quadratmeter, weshalb diese Variable einfach festgelegt werden kann. Für den Namen und die Beschreibung einer Komposition muss zunächst festgestellt werden, ob es sich um ein einschichtiges oder mehrschichtiges Bauteil handelt. Je nach Art des Bauteils sind die relevanten Informationen an unterschiedlichen Knoten im IFC gespeichert.

Bauteilschichten sind, wie zuvor beschrieben, mit einem *IfcAggregate* verknüpft<sup>53</sup>. Die Bauteilschichten werden in der Reihenfolge gespeichert, in der sie in der Wand vorkommen. Somit kann durch den Container iteriert werden, wobei angenommen wird, dass die Reihenfolge der Einträge im Container auch die Reihenfolge der Schichten im Bauteil ist. Das Attribut der Schichtdicke eines mehrschichtigen Bauteils wird ArchiCAD- und sprachspezifisch abgespeichert. Deshalb muss mithilfe des Dictionarys in der Konfigurationsdatei der sprachspezifische Name des Attributs abgeglichen werden. Bei einschichtigen Bauteilen kann die Gesamtdicke verwendet werden, die sprachunabhängig gespeichert wird.

<sup>53</sup>Figure 14

Die Schicht, dargestellt durch *IfcElementPart*, kann ab diesem Punkt wie ein einschichtiges *IfcElement* behandelt werden. Das Material mit allen Eigenschaften kann auf die gleiche Weise abgerufen werden.



**Figure 14: Beziehungen von einer IfcWall zur Schichtdicke und der ÖKOBAUDAT UUID eines IfcElementPart als Graph.**

Dataclasses in Python sind spezialisierte Klassen, die hauptsächlich zur Speicherung von Daten verwendet werden. Sie reduzieren den *Boilerplate-Code*, der normalerweise mit der Initialisierung, Repräsentation und Gleichheit von Objekten verbunden ist, durch automatische Generierung dieser Methoden<sup>54</sup>. Dataclasses sind nützlich für einfache, containerartige Strukturen, in denen hauptsächlich Attribute gespeichert werden sollen. Diese werden nun verwendet, um gesuchte Attribute eines IFC-Elements und deren Schichten zu speichern oder Standardvariablen zu definieren, falls diese nicht gefunden werden. Alle Schichten eines Elements werden als Liste an das Objekt angehängt.

Diese beiden Klassen *IfcElement* und *IfcLayer* wurden wie folgt definiert:

```

1 @dataclass
2 class IfcElement:
3     """Class for saving Elementsattributes of Elements"""
4
5     uuid: str
6     din276Code: int
7     refUnit: str
8     comp_name: str
9     layers: list
10    comp_description: str | None = ""

```

```

1 @dataclass
2 class IfcLayer:
3     """Class for keeping track of Layerinformation of a Element"""
4
5     layer_material: str
6     layer_size: float
7     layer_position: int
8     layer_uuid: str
9     layer_lifeTime: int
10    layer_lifeTimeDelay: int | None = 0
11    calcLca: bool | None = True
12    isExtant: bool | None = False

```

Jeder einzigartige Wandaufbau wird als Objekt gespeichert und weitergegeben um daraus ein eLCA-Objekt zu bauen.

<sup>54</sup>Python Software Foundation (2024)

## 4.3 Erstellung von eLCA-XML-Dateien aus Bauteil-Objekten

Die Objekte, die aus der Analyse der IFC erstellt wurden, werden nun an sehr ähnliche eLCA-Klassen übergeben, um daraus erneut Objekte zu erstellen. Die gleichen Objekte wiederzuverwenden wäre theoretisch möglich gewesen, doch indem die einzelnen Arbeitsschritte ihre eigenen Klassen erhalten, wird für zukünftige Erweiterungen und Kompatibilität gesorgt.

```

1 @dataclass
2 class ElcaElement:
3     """Class for keeping track of SubElement Values of ElementTree"""
4
5     uuid: str
6     din276Code: int
7     refUnit: str
8     name: str
9     description: str | None = ""
10    elca_eol: str | None = ""
11    elca_recycling: str | None = ""
12    elca_seperation: str | None = ""
13    elca_rW: str | None = ""
14    elca_uValue: str | None = ""
15    quantity: str | None = "1"

```

```

1 @dataclass
2 class ElcaComponent:
3     """Class for keeping track of Component Values of Components Tree
4
5     processConfigUuid: str
6     processConfigName: str
7     layerPosition: str
8     layerSize: str
9     lifeTime: str
10    lifeTimeDelay: str | None = "0"
11    isLayer: str | None = "true"
12    calcLca: str | None = "true"
13    isExtant: str | None = "false"
14    layerAreaRatio: str | None = "1.0"
15    layerLength: str | None = "1.0"
16    layerWidth: str | None = "1.0"

```

Alle Parameter eines Objektes der Klasse *ElcaElement* werden in die XML-Vorlagedatei geschrieben. Die Parameter von Objekten der Klasse *Elca-Component* werden als XML-String generiert und an die übergeordneten Komponenten angehängt. Danach wird der gesamte XML-String, der mit bauteilbezogenen Informationen gefüllt ist, als neue XML-Datei gespeichert und mit der definierten UUID des Bauteils benannt.

Die generierten XML-Dateien können in eLCA problemlos importiert werden und für die Ökobilanzierung in jedem beliebigen Projekt verwendet werden.

## 4.4 Anwendung als Command-Line-Tool

Ein Command-Line-Tool ist ein Programm, das über eine Textschnittstelle bedient wird und Befehle über eine Kommandozeile entgegennimmt. Sie eignet sich besonders gut, da sie oft weniger Ressourcen benötigen als grafische Benutzeroberflächen, leicht zu skripten und zu automatisieren sind und in vielen Systemumgebungen verfügbar sind. Zudem ermöglichen sie präzise und schnelle Interaktionen mit dem System, besonders für erfahrene Benutzer<sup>55</sup>.

Das Command-Line-Tool kann entweder mit der *pyproject.toml* des GitHub-Repositories<sup>56</sup> oder dem Source-Code der Thesis über *pip* installiert werden. Dabei werden alle Abhängigkeiten automatisch installiert. Die Referenzierung auf den Export-Ordner für die XML-Dateien und auf die IFC-Datei sowie die Angabe der verwendeten Sprache erfolgen mittels Parsing-Argumenten. Standardmäßig ist die Sprache auf *GERMAN* gesetzt. Eine detaillierte Anleitung zur Installation und Nutzung des Tools findet sich auf GitHub. Das Tool kann sowohl auf Servern als auch auf Heimrechnern installiert werden und sollte für jede EDV-Abteilung eines Architekturbüros problemlos in den Workflow zu implementieren sein.

<sup>55</sup>The Linux Documentation Project (2024)

<sup>56</sup>Bergmann (2024)

# 5 Fazit und Ausblick auf zukünftige Forschungsansätze

## 5.1 Analyse der Ergebnisse

Das entwickelte Programm liefert teilautomatisiert XML-Bauteilvorlagen für das LCA-Tool eLCA aus beliebigen IFC4-Dateien, die aus ArchiCAD exportiert wurden. Diese Automatisierung ermöglicht es, Bauteilinformationen effizienter und fehlerfreier in das LCA-Tool zu übertragen. Das Projekt wurde so konzipiert und publiziert, dass es zur Weiterentwicklung anregen soll, um den gesamten Prozess weiter zu optimieren und zu erweitern.

Derzeit ist das Programm darauf beschränkt, Bauteilvorlagen von *IfcWalls* zu erstellen. *IfcWalls* (also Wände) sind jedoch nur für einen Teil der Gesamtökobilanz eines Gebäudes verantwortlich. Um eine umfassendere und genauere Ökobilanz zu erstellen, wäre es sinnvoll, den Prozess auf alle *IfcElements*, wie *IfcSlabs* (Decken), *IfcRoofs* (Dächer), *IfcWindows* (Fenster) und andere relevante Bauteile zu erweitern. Die Methode zur Abfrage und Extraktion von Eigenschaften ist für *IfcSlabs* und andere Elemente ähnlich wie bei *IfcWalls*, sodass der Erweiterungsaufwand in der Praxis durch eine Anpassung des bestehenden Codes realisierbar ist.

Zusätzlich besteht die Möglichkeit, das entwickelte Tool mit dem Source-Code von Jil Schneider zu kombinieren. Eine solche Zusammenführung könnte den Prozess des IFC-Imports in eLCA vollständig automatisieren. Dadurch würde nicht nur die Extraktion und Aufbereitung der Bauteilschichten,

sondern auch die direkte Integration der Daten in eLCA ermöglicht, was den gesamten Arbeitsablauf erheblich vereinfachen und beschleunigen könnte.

Der derzeitige Prozess zur Extraktion der Schichten und deren Eigenschaften aus ArchiCAD und deren Verpackung in XML-Bauteilvorlagen ist vollständig automatisiert. Allerdings gibt es noch einen manuellen Schritt, der die UUIDs (Universally Unique Identifiers) der ÖKOBAUDAT-Datenbank in die Eigenschaften der Bauteile einfügt. Diese UUIDs sind notwendig, um die Materialien und deren Umweltauswirkungen korrekt zuzuordnen.

Es gibt zwei Hauptoptionen, um diesen Schritt zu automatisieren:

#### **Direkter Bezug von eLCA-Baustoffen:**

Eine Möglichkeit wäre, die Baustoffdaten direkt von der eLCA-Webseite zu beziehen. Dies könnte durch eine API-Integration erfolgen, bei der die notwendigen Baustoffinformationen direkt in die XML-Bauteilvorlagen integriert werden.

#### **Matchingverfahren:**

Eine andere Möglichkeit besteht darin, ein Matchingverfahren zu entwickeln, das die Baustoffe in der IFC-Datei mit den entsprechenden Baustoffen in der ÖKOBAUDAT-Datenbank abgleicht und automatisch die richtigen UUIDs zuordnet. Dieses Verfahren würde eine Art Übersetzungsmechanismus darstellen, der die im BIM-Modell verwendeten Baustoffe mit den standardisierten Materialien der ÖKOBAUDAT-Datenbank verknüpft.

Beide Optionen haben das Potenzial, den Prozess weiter zu automatisieren und die Genauigkeit der Ökobilanzierung zu erhöhen. Der Einsatz solcher Technologien könnte die Effizienz und Präzision der LCA-Analyse im Bauwesen erheblich verbessern und dazu beitragen, nachhaltigeres Bauen zu fördern. Durch die kontinuierliche Weiterentwicklung und Integration neuer Funktionen kann dieses Projekt einen bedeutenden Beitrag zur Verbesserung der digitalen Bauwerksdokumentation und der nachhaltigen Bauplanung leisten.

## **5.2 eLCA Baustoffe in ArchiCAD importieren**

Auf der Webseite von eLCA befindet sich eine umfassende Datenbank, die alle von der ÖKOBAUDAT unterstützten Baustoffe enthält. Diese Datenbank stellt eine zentrale Ressource für die Erstellung von Bauteilvorlagen dar, da sie alle notwendigen Informationen für die spätere Generierung der XML-Bauteilvorlagen enthält. Die ÖKOBAUDAT-Datenbank bietet detaillierte Informationen zu jedem Baustoff, einschließlich der Umweltkenwerte, die für die Erstellung präziser Ökobilanzen notwendig sind<sup>57</sup>.

<sup>57</sup>Figure 15

**Balkenschichtholz (Durchschnitt DE) BAUSTOFF**

General		lifecycle																												
Name*	Balkenschichtholz (Durchschnitt DE)																													
Übersetzung	Glued solid timber (German average) [en]																													
Notes	<input type="text"/> Visible to users <input checked="" type="checkbox"/> 4108 Baustoff <input type="checkbox"/>																													
Thickness mm	Factor Hs / Hi	Lamda (λ) W/mK	<input type="text"/>																											
AVV	<input type="text"/>  Holz Diele (default) <input type="button" value="▼"/>																													
<b>useful lives</b> General information on the useful life Min. a Information about the minimum useful life 50 <input type="text"/> Average a Information about the average useful life Max. a Information about the maximum useful life  <b>Hatching</b>																														
<b>Unterstützte Datenbanken und Umrechnungsfaktoren</b> <table border="1"> <thead> <tr> <th></th> <th>Quantitative Referenz</th> <th>kg / m³</th> </tr> </thead> <tbody> <tr> <td>OeKOBAUDAT_2016_I_A1</td> <td>1.000 m³</td> <td>500.360</td> </tr> <tr> <td>ÖBD Release Stand 21-01-2019</td> <td>1.000 m³</td> <td>500.360</td> </tr> <tr> <td>ÖBD_2019_III</td> <td>1.000 m³</td> <td>500.360</td> </tr> <tr> <td>ÖBD_2020_II_A1</td> <td>1.000 m³</td> <td>500.360</td> </tr> <tr> <td>ÖBD_2021_II_A1</td> <td>1.000 m³</td> <td>500.360</td> </tr> <tr> <td>Ökobilanzierung-Rechenwerte 2023 V1.3</td> <td>1.000 m³</td> <td>500.360</td> </tr> <tr> <td>ÖBD_2023_I_A1</td> <td>1.000 m³</td> <td>500.360</td> </tr> <tr> <td>ÖBD_2023_I_A2</td> <td>1.000 m³</td> <td>500.360</td> </tr> </tbody> </table>					Quantitative Referenz	kg / m³	OeKOBAUDAT_2016_I_A1	1.000 m³	500.360	ÖBD Release Stand 21-01-2019	1.000 m³	500.360	ÖBD_2019_III	1.000 m³	500.360	ÖBD_2020_II_A1	1.000 m³	500.360	ÖBD_2021_II_A1	1.000 m³	500.360	Ökobilanzierung-Rechenwerte 2023 V1.3	1.000 m³	500.360	ÖBD_2023_I_A1	1.000 m³	500.360	ÖBD_2023_I_A2	1.000 m³	500.360
	Quantitative Referenz	kg / m³																												
OeKOBAUDAT_2016_I_A1	1.000 m³	500.360																												
ÖBD Release Stand 21-01-2019	1.000 m³	500.360																												
ÖBD_2019_III	1.000 m³	500.360																												
ÖBD_2020_II_A1	1.000 m³	500.360																												
ÖBD_2021_II_A1	1.000 m³	500.360																												
Ökobilanzierung-Rechenwerte 2023 V1.3	1.000 m³	500.360																												
ÖBD_2023_I_A1	1.000 m³	500.360																												
ÖBD_2023_I_A2	1.000 m³	500.360																												

**Figure 15: Browseransicht: Baustoff in eLCA.**

Durch die Verwendung einer API-Schnittstelle könnten die relevanten Baustoffinformationen, wie zum Beispiel die Materialkennwerte und Umweltindikatoren, aus der ÖKOBAUDAT-Datenbank extrahiert und in eine XML-Datei überführt werden<sup>58</sup>. Diese XML-Datei könnte dann bei der Erstellung einer neuen ArchiCAD-Datei in die BIM-Software importiert werden, um dort direkt zur Verfügung zu stehen.

<sup>58</sup>Lambertz et al. (2019)

Ein besonders interessanter Aspekt der eLCA-Webseite ist, dass sie nicht nur die technischen Daten der Baustoffe, sondern auch Schraffuren bereitstellt. Schraffuren sind grafische Muster, die in Bauzeichnungen verwendet werden, um unterschiedliche Materialien darzustellen. Die Möglichkeit, diese Schraffuren ebenfalls zu übernehmen, würde sicherstellen, dass die generierten Baustoffe in ArchiCAD-Schnitten und -Ansichten korrekt und konsistent angezeigt werden. Dies verbessert nicht nur die visuelle Darstellung, sondern auch die Genauigkeit der Bauwerksdokumentation.

Trotz der vielen Vorteile gibt es einige potenzielle Limitierungen bei dieser Methode. Eine wichtige Unsicherheit betrifft die Kompatibilität mit zukünftigen Versionen von ArchiCAD. Software-Updates können Änderungen in der Datenstruktur oder den Importfunktionen mit sich bringen, was dazu führen könnte, dass der aktuelle Web-Scraper oder das Importskript angepasst werden müssen. Diese Abhängigkeit von der Softwareversion könnte zusätzlichen Wartungsaufwand bedeuten.

Ein weiterer Punkt, der beachtet werden muss, ist die fehlende Information über die Oberflächentexturen der Baustoffe in der ÖKOBAUDAT-Datenbank. Während Schraffuren eine wichtige Rolle in der grafischen Darstellung spielen, sind Oberflächentexturen essenziell für realistische

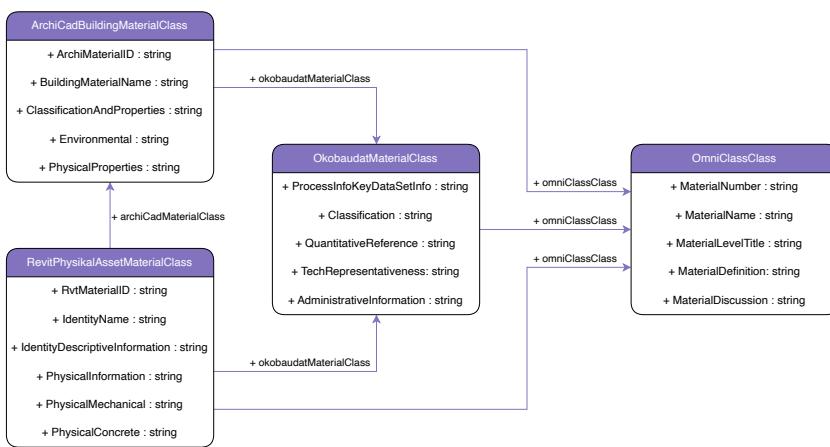
Visualisierungen und Renderings. Ohne diese Texturen könnten die importierten Baustoffe in 3D-Visualisierungen unvollständig oder ungenau dargestellt werden. Da die Baustoffe möglichst früh in das BIM-Projekt importiert und verwendet werden, müssten fehlende Texturen von anderen Plattformen gezogen werden oder bereitgestellte, generischen Texturen vor oder nach dem Import eingefügt werden. Dennoch gäbe es auch hier die Schwierigkeit das für zukünftige Versionen und Bauteile fortlaufend zu aktualisieren.

Zusammengefasst bietet die Nutzung der eLCA-Datenbank in Verbindung mit einer API eine vielversprechende Möglichkeit, den Prozess der Baustoffintegration in ArchiCAD zu automatisieren. Dies könnte die Effizienz und Genauigkeit der Erstellung von Bauteilvorlagen erheblich verbessern. Die Herausforderungen und Limitierungen dieser Methode, insbesondere die Kompatibilität mit zukünftigen Softwareversionen und die fehlenden Oberflächentexturen, müssen jedoch berücksichtigt und durch entsprechende Lösungen adressiert werden. Eine kontinuierliche Weiterentwicklung und Anpassung des Systems wird notwendig sein, um den größtmöglichen Nutzen aus dieser innovativen Integration zu ziehen.

### 5.3 Baustoffzuweisung mit Matching-Verfahren

Das in der Arbeit von Bangalu Apollos Yohanna und Jakob Beetz vorgestellte Modell zur Integration von BIM und LCA zielt darauf ab, die Material- und Produktinformationen zwischen BIM- und LCA-Datenbanken abzugleichen und zu verknüpfen<sup>59</sup>. Hierbei kommt ein semantischer Matching-Algorithmus zum Einsatz, der die Attribute der BIM-Objekte mit denen der LCA-Datenbankeinträge vergleicht. Der Algorithmus nutzt spezifische technische und physische Eigenschaften, um Übereinstimmungen von Betonbaustoffen zu identifizieren.

<sup>59</sup> Bangalu and Beetz (n.d.)



**Figure 16: UML-Klassendiagramm der relevanten Datenpunkte für den Abgleich von Betonmaterialien/Produkten<sup>60</sup>.**

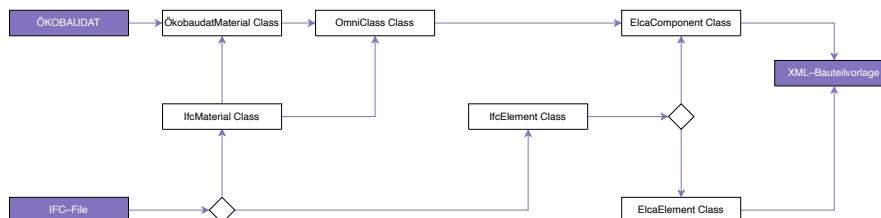
<sup>60</sup> Bangalu and Beetz (n.d.)

Mit der Verwendung eines trainierten Large Language Models (LLM) könnte, dieser Prozess noch auf weitere Baustoffe ausgeweitet werden. Na-

men der ÖKOBAUDAT-Materialien könnten in Mengen von ähnlichen Materialien identifizieren werden, deren Namen den BIM-Materialien ähneln.

Der Abgleich läuft in zwei Schritten ab: Zunächst wird eine Vorselektion der Materialien anhand der Namensähnlichkeit durchgeführt. Danach erfolgt ein detaillierter Vergleich der technischen Eigenschaften wie Dichte, Wärmeleitfähigkeit und Wärmekapazität. Der nearest neighbour-Algorithmus wird verwendet, um das Material zu finden, dessen Eigenschaften am besten mit denen des BIM-Materials übereinstimmen. Das bedeutet, dass das Material ausgewählt wird, bei dem die Summe der Unterschiede in den Eigenschaften minimal ist. Der Ablauf der relevanten Datenpunkte für den Ableich wird in Figure 16 beschrieben.

Die Methode ist skalierbar und kann potenziell auf verschiedene Bauprojekte und -größen angewendet werden. Der manuelle Eingriff in den Prozess wäre minimal. Das LLM würde sich auch auf kommende Versionen und sich verändernden Baustoffen anpassen. Dies macht sie besonders attraktiv für große Bauunternehmen und Projekte mit umfangreichen Materialdatenbanken. Für zukünftige Forschungsansätze wäre es interessant diese Algorithmen in das Modell dieser Arbeit zu integrieren. Ein möglicher Ansatz zur Integration anhand des Datenmodells aus Figure 16 ist in Figure 17 beschrieben.



**Figure 17: UML-Diagramm mit möglicher Integration der beiden Projekte.**

## Literaturverzeichnis

- Archicad klassifizierungen und kategorien in ifc.* (2024, June 23). Graphisoft. [https://help.graphisoft.com/AC/23/GER/\\_AC23\\_Help/115\\_IFC/115\\_IFC-3.htm](https://help.graphisoft.com/AC/23/GER/_AC23_Help/115_IFC/115_IFC-3.htm)
- Bangalu, A. Y., & Beetz, J. (n.d.). *Bim-lca integration – a conceptual database alignment and conformity model*, RWTH Aachen University.
- Baumann, H., & Tillman, A.-M. (2004). The Hitch Hiker's Guide to LCA: An Orientation in Life Cycle Assessment Methodology and Application. *Studentlitteratur*.
- Bergmann, J. (2024). Archicad2elca\_comps. [https://github.com/johnzoki/ArchiCAD2eLCA\\_comps](https://github.com/johnzoki/ArchiCAD2eLCA_comps)
- BMUB. (2017). *Ökobaudat 2017: Ökologische bewertungsdaten für das bauwesen*. Bundesministerium für Umwelt, Naturschutz, Bau und Reaktorsicherheit.
- Borrman, A., König, M., Koch, C., & Beetz, J. (Eds.). (2021). *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (2. Auflage). Springer Fachmedien Wiesbaden. <https://doi.org/10.1007/978-3-658-33361-4>
- Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2008, November 26). *Extensible markup language (xml) 1.0 (fifth edition)*. World Wide Web Consortium (W3C). <https://www.w3.org/TR/REC-xml/>
- Bribian, I. Z., Capilla, A. V., & Usón, A. A. (2011). Life cycle assessment in buildings: State-of-the-art and simplified lca methodology as a complement for building certification. *Building and Environment*, 44.
- BuildingSMART. (2024, June 23). *Ifc 4.3.2 documentation*. <https://ifc43-docs.standards.buildingsmart.org>
- Cole, R. J. (2005). Building environmental assessment methods: Redefining intentions and roles. *Building Research & Information*, 33.
- DGNB. (2020). *Elca - das online-tool für ökobilanzierung im bauwesen*. Deutsche Gesellschaft für Nachhaltiges Bauen.
- Din 276 [Deutsche Norm]. (2018).
- Din en 15978 [Deutsche Norm]. (2011).
- Din en iso 14040 [Deutsche Norm]. (2021).
- Dodoo, A., Gustavsson, L., & Sathre, R. (2014). Lifecycle carbon implications of conventional and low-energy multi-storey timber building systems. *Energy and Buildings*, 82. <https://www.sciencedirect.com/science/article/pii/S0378778814005167>
- Elca online handbuch.* (2024, June 22). <https://www.energieberatung.tv/Handbuch/eLCA%20Online%20Handbuch.html>
- Göbels, A. (2020). Klassifizierung von bauteilen auf grundlage der din spec 91400 in archicad.
- IfcOpenShell. (2024). *Ifcopenshell 0.7.0 documentation*. <https://docs.ifcopenshell.org>
- ILCD. (2010). *Nternational reference life cycle data system (ilcd) handbook - general guide for life cycle assessment - detailed guidance*. European Commission - Joint Research Centre - Institute for Environment; Sustainability.
- Kibert, C. J. (2016). *Sustainable construction: Green building design and delivery* (4. Auflage). John Wiley & Sons.
- Lambertz, M., Theißen, S., Höper, J., Wimmer, R., Meins-Becker, A., & Zibell, M. (2019). Ökobilanzierung und bim im nachhaltigen bauen.
- Python Software Foundation. (2024). *Dataclasses – data classes*. <https://docs.python.org/3/library/dataclasses.html>
- Rössig, S. (2014). *Elca starter-handbuch*.

Schneider, J. (2021). *Gebäude als materialressource - openbim in der ökobilanzierung* [Master's thesis, RWTH Aachen University].

The Linux Documentation Project. (2024). *Introduction to the command line*. <https://tldp.org>.

Wernet, G., Bauer, C., Steubing, B., Reinhard, J., Moreno-Ruiz, E., & Weidema, B. (2016). The ecoinvent database version 3 (part i): Overview and methodology. *The International Journal of Life Cycle Assessment*, 21.

# Appendix

## \_\_main\_\_.py

Datei, die ausgeführt wird beim Start des Programms.

```

1 from archicad2elca_comps.__init__ import main
2
3 if __name__ == "__main__":
4     main()

```

## \_\_init\_\_.py

Datei mit *main*-Funktion, die als erstes vom Programm aufgerufen wird.

```

1 import argparse
2 from archicad2elca_comps.localization import Lang
3 from archicad2elca_comps.ifc_extractor import ifc_extractor
4 from archicad2elca_comps.xml_builder import xml_builder
5
6
7 def main():
8     parser = argparse.ArgumentParser()
9     parser.add_argument(
10         "--language",
11         "-l",
12         type=str,
13         required=False,
14         dest="lang",
15         default="GERMAN",
16         help="Set the language",
17     )
18     parser.add_argument(
19         "--ifcfile",
20         "-f",
21         type=str,
22         required=True,
23         dest="ifcfile",
24         help="Path to your IFC-File",
25     )
26     parser.add_argument(
27         "--exportfolder",
28         "-e",
29         type=str,
30         required=True,
31         dest="exportfolder",
32         help="Path to your ExportFolder",
33     )
34
35     args = parser.parse_args()
36
37     Lang(args.lang)
38     ifcfile_path = args.ifcfile
39     exportfolder_path = args.exportfolder
40
41     print(
42         """ARCHICAD2ELCA_COMPS
43         AUTOR:
44             name: Bergmann, John Herbert
45             email: john.bergmann@rwth-aachen.de
46         DESCRIPTION:
47             A Python project that imports archicad composites out of ifc and archicad attributes.xml

```

```

    and exports eLCA composites for easy LCA.
48 CLASSIFIERS:
49     Programming Language :: Python :: 3
50     License :: OSI Approved :: MIT License
51     Operating System :: OS Independent
52     Development Status :: 1 - Planning
53
54     STARTING ...
55     """
56 )
57
58 extracted_elements = ifc_extractor(ifcfile_path=ifcfile_path)
59 xml_builder(exportfolder_path=exportfolder_path, elements=extracted_elements)

```

## config.py

Konfigurationsdatei.

```

1 # SUPPORTED LANGUAGES
2 # Set Archicad Language specific Variables
3 languages = {
4     "GERMAN": {
5         "AC_multilayer_pset_name": "Mehrschichtige Bauteile",
6         "AC_quantity_thickness_name": "Schichtdicke",
7     },
8     "ENGLISH": {
9         "AC_multilayer_pset_name": "Composite Structure",
10        "AC_quantity_thickness_name": "Component Thickness",
11    },
12 }

```

## ifc\_extractor.py

Liest und speichert gesuchte Informationen aus IFC-Dateien.

```

1 import os
2 import ifcopenshell
3 from dataclasses import dataclass
4 from enum import Enum
5 import uuid
6 from archicad2elca_comps.localization import Lang
7
8 AC_lang_vars = Lang()
9
10
11 class CompType(Enum):
12     SINGLELAYER = 1
13     MULTILAYER = 2
14
15
16 class FaultyElementAttributeError(Exception):
17     def __init__(self, message, errors):
18         super().__init__(message)
19         self.errors = errors
20
21
22 class IFCExportError(Exception):
23     def __init__(self, message, errors):
24         super().__init__(message)
25         self.errors = errors
26

```

```

27
28 @dataclass
29 class IfcElement:
30     """Class for saving Elementsattributes of Elements"""
31
32     uuid: str
33     din276Code: int
34     refUnit: str
35     comp_name: str
36     layers: list
37     comp_description: str | None = ""
38
39
40 @dataclass
41 class IfcLayer:
42     """Class for keeping track of Layerinformation of a Element"""
43
44     layer_material: str
45     layer_size: float
46     layer_position: int
47     layer_uuid: str
48     layer_lifeTime: int
49     layer_lifeTimeDelay: int | None = 0
50     calcLca: bool | None = True
51     isExtant: bool | None = False
52
53
54 def get_uuid() -> str:
55     """
56     Returns a random uuid
57     """
58     return str(uuid.uuid4())
59
60
61 def property_finder(ifc_element, property_set, property_name) -> str:
62     """
63     Returns attribute values from RelatingPropertyDefinitions
64     """
65     for s in ifc_element.IsDefinedBy:
66         if (
67             hasattr(s, "RelatingPropertyDefinition")
68             and s.RelatingPropertyDefinition.Name == property_set
69         ):
70             if hasattr(s.RelatingPropertyDefinition, "HasProperties"):
71                 for v in s.RelatingPropertyDefinition.HasProperties:
72                     if v.Name == property_name:
73                         return v.NominalValue.wrappedValue
74             elif hasattr(s.RelatingPropertyDefinition, "Quantities"):
75                 for v in s.RelatingPropertyDefinition.Quantities:
76                     if v.Name == property_name:
77                         for attr, value in vars(v).items():
78                             if attr.endswith("Value"):
79                                 return value
80
81
82     return None
83
84
85 def get_din276Code(ifc_element):
86     """
87     Returns din276Codes of IfcElements by checking Pset information.
88
89     Parameters:
90         ifc_element:
91             IfcElement in IFC-Model. In this case a wall
92
93     Compare:
94         filename: KG.py;
95         author: Jil Schneider
96         project: Masterthesis Architektur
97         title: Building as Material Resource - Life Cycle Assessment with OpenBIM
98         date: 2021/02/19

```

```

97 """
98 reference = property_finder(ifc_element, "Pset_WallCommon", "Reference")
99 isExternal = property_finder(ifc_element, "Pset_WallCommon", "IsExternal")
100 isLoadbearing = property_finder(ifc_element, "Pset_WallCommon", "Loadbearing")
101 isExtendToStructure = property_finder(
102     ifc_element, "Pset_WallCommon", "ExtendToStructure"
103 )
104 if isExternal is not None:
105     if isLoadbearing is not None:
106         if isExtendToStructure is not None:
107             if isLoadbearing and isExternal and not isExtendToStructure:
108                 return 331
109                 # Tragende Aussenwaende
110             elif not isLoadbearing and isExternal and not isExtendToStructure:
111                 return 332
112                 # Nichttragende Aussenwaende
113             elif not isLoadbearing and isExternal and isExtendToStructure:
114                 return 335
115                 # Aussenwandbekleidung, aussen
116             elif (
117                 not isLoadbearing
118                 and not isExternal
119                 and (isExtendToStructure and referenceForExternalWall in reference)
120             ):
121                 return 336
122                 # Aussenwandbekleidung, innen
123             elif isLoadbearing and not isExternal and not isExtendToStructure:
124                 return 341
125                 # Tragende Innenwaende
126             elif not isLoadbearing and not isExternal and not isExtendToStructure:
127                 return 342
128                 # Nichttragende Innenwaende
129             elif (
130                 not isLoadbearing
131                 and not isExternal
132                 and (isExtendToStructure and referenceForInternalWall in reference)
133             ):
134                 return 345
135                 # Innenwandbekleidung
136             elif not isLoadbearing and not isExternal and isExtendToStructure:
137                 return 000
138             else:
139                 raise IFCExportError("ExtantToStructure is missing in Pset_WallCommon")
140         elif isExternal:
141             return 330
142             # Aussenwaende/Vertikale Baukonstruktionen, aussen
143         elif not isExternal:
144             return 340
145             # Innenwaende/Vertikale Baukonstruktionen, innen
146     else:
147         raise IFCExportError("Loadbearing is missing in Pset_WallCommon")
148 else:
149     return 300
150     # Bauwerk Baukonstruktionen
151
152 def get_material_info(m, ifc_material):
153 """
154 Gets specific values in "AC_Pset_MaterialCustom" and name of a given material.
155
156 Parameters:
157     m:
158         IFC-Model
159     ifc_material:
160         Given IfcMaterial of IfcElement or IfcElementPart
161
162 layer_uuid = None
163 layer_lifeTime = None
164 layer_lifeTimeDelay = None
165 calcLca = None

```

```

167     isExtant = None
168     material_inverse = m.get_inverse(ifc_material)
169     for i in material_inverse:
170         if i.Name == "AC_Pset_MaterialCustom":
171             for prop in i.Properties:
172                 if prop.is_a("IfcPropertySingleValue"):
173                     if prop.Name == "OBD_uuid":
174                         layer_uuid = prop.NominalValue[0]
175                     elif prop.Name == "OBD_lifeTime":
176                         layer_lifeTime = prop.NominalValue[0]
177                     elif prop.Name == "OBD_lifeTimeDelay":
178                         layer_lifeTimeDelay = prop.NominalValue[0]
179                     elif prop.Name == "OBD_calcLca":
180                         calcLca = prop.NominalValue[0]
181                     elif prop.Name == "OBD_isExtant":
182                         isExtant = prop.NominalValue[0]
183             break
184
185     ifc_material_name = ifc_material.Name
186     return (
187         ifc_material_name,
188         layer_uuid,
189         layer_lifeTime,
190         layer_lifeTimeDelay,
191         calcLca,
192         isExtant,
193     )
194
195
196 def multi_layer(m, ifc_rel_aggregates):
197     """
198     Creates Objects from IfcLayer classes from Elements with multiple Layers.
199
200     Parameters:
201         m:
202             IFC-Model
203         ifc_rel_aggregates:
204             Set of IfcElementParts of one IfcElement
205     """
206     first_aggregate = ifc_rel_aggregates[0]
207     element_comp = first_aggregate[5]
208     comp_layer_list = []
209     for layer_number, ifc_element_part in enumerate(element_comp, 1):
210         ifc_material_name = None
211         ifc_material = ifcopenshell.util.element.get_material(ifc_element_part)
212         if ifc_material is not None and ifc_material.is_a("IfcMaterial"):
213             (
214                 ifc_material_name,
215                 layer_uuid,
216                 layer_lifeTime,
217                 layer_lifeTimeDelay,
218                 calcLca,
219                 isExtant,
220             ) = get_material_info(m, ifc_material)
221         if ifc_material_name is None:
222             raise FaultyElementAttributeError("Material_name is not defined")
223
224         width = None
225         quant = ifcopenshell.util.element.get_pset(
226             ifc_element_part, "Component Quantities"
227         )
228         if quant:
229             width = quant.get(AC_lang_vars.AC_quantity_thickness_name)
230         if width is not None:
231             l = IfcLayer(
232                 layer_material=ifc_material_name,
233                 layer_size=width,
234                 layer_position=layer_number,
235                 layer_uuid=layer_uuid,
236                 layer_lifeTime=int(layer_lifeTime),

```

```

237     layer_lifeTimeDelay=(
238         int(layer_lifeTimeDelay)
239         if layer_lifeTimeDelay is not None
240         else IfcLayer.layer_lifeTimeDelay
241     ),
242     calcLca=calcLca if calcLca is not None else IfcLayer.calcLca,
243     isExtant=isExtant if isExtant is not None else IfcLayer.isExtant,
244   )
245   comp_layer_list.append(l)
246 if comp_layer_list:
247   return comp_layer_list
248 else:
249   raise FaultyElementAttributeError("Empty Layer List")
250
251
252 def wall_single_layer(m, ifc_element):
253   """
254   Creates one Object from IfcLayer class from Element with just one Layers.
255
256   Parameters:
257     m:
258       IFC-Model
259     ifc_element:
260       Given IfcElement
261   """
262   w_pset = ifcopenshell.util.element.get_psets(ifc_element)
263   pset_Qto_WallBaseQuantities = w_pset.get("Qto_WallBaseQuantities")
264   width = None
265   if pset_Qto_WallBaseQuantities:
266     width = pset_Qto_WallBaseQuantities.get("Width")
267   if width is None:
268     raise FaultyElementAttributeError("Width is not defined")
269
270   ifc_material_name = None
271   ifc_material = ifcopenshell.util.element.get_material(ifc_element)
272   if ifc_material is not None and ifc_material.is_a("IfcMaterial"):
273     (
274       ifc_material_name,
275       layer_uuid,
276       layer_lifeTime,
277       layer_lifeTimeDelay,
278       calcLca,
279       isExtant,
280     ) = get_material_info(m, ifc_material)
281
282   if ifc_material_name is None:
283     raise FaultyElementAttributeError("Material_name is not defined")
284
285   comp_layer_list = [
286     IfcLayer(
287       layer_material=ifc_material_name,
288       layer_size=width,
289       layer_position=1,
290       layer_uuid=layer_uuid,
291       layer_lifeTime=int(layer_lifeTime),
292       layer_lifeTimeDelay=(
293         int(layer_lifeTimeDelay)
294         if layer_lifeTimeDelay is not None
295         else IfcLayer.layer_lifeTimeDelay
296       ),
297       calcLca=calcLca if calcLca is not None else IfcLayer.calcLca,
298       isExtant=isExtant if isExtant is not None else IfcLayer.isExtant,
299     )
300   ]
301   comp_name = ifc_material_name + " " + str(width * 100) + "cm"
302   return comp_name, comp_layer_list
303
304
305 def get_comp_type(m, ifc_element):
306   """

```

```

307 Decides if given IfcElement is a multilayered Element or singlelayered.
308
309 Parameters:
310     m:
311         IFC-Model
312     ifc_element:
313         Given IfcElement
314 """
315 element_ref = m.get_inverse(ifc_element)
316 ifc_rel_aggregates = [rel for rel in element_ref if rel.is_a("IfcRelAggregates")]
317 if ifc_rel_aggregates:
318     AC_pset = ifcopenshell.util.element.get_pset(ifc_element, "ArchiCADProperties")
319     comp_name = None
320     if AC_pset:
321         if AC_pset.get(AC_lang_vars.AC_multilayer_pset_name):
322             comp_name = AC_pset.get(AC_lang_vars.AC_multilayer_pset_name)
323     else:
324         raise FaultyElementAttributeError(
325             f"No Pset named {AC_lang_vars.AC_multilayer_pset_name} in ArchiCADProperties"
326         )
327     if comp_name:
328         comp_layer_list = multi_layer(m, ifc_rel_aggregates)
329         return comp_name, comp_layer_list, CompType.MULTILAYER
330     else:
331         raise FaultyElementAttributeError("No Compname in ArchiCADProperties")
332 else:
333     # No IfcRelAggregates found for Wall
334     return (*wall_single_layer(m, ifc_element), CompType.SINGLELAYER)
335
336
337 def ifc_extractor(ifcfile_path):
338 """
339 Creates Objects from IfcElement class from Elements and collects all neccessary data to do so.
340 Returns these Objects to '__init__.py' to later use them in the xml_builder.
341
342 Parameters:
343     ifcfile_path:
344         Given path to IFC4-file, that is to be used for extracting comps
345 """
346 m = ifcopenshell.open(ifcfile_path)
347 walls = m.by_type("IfcWall")
348 element_dict = {}
349 for wall in walls:
350     try:
351         comp_name, layer_list, comp_type = get_comp_type(m, wall)
352     except FaultyElementAttributeError:
353         continue
354     uuid = get_uuid()
355     din276Code = get_din276Code(wall)
356     refUnit = "m2"
357
358     element_dict.setdefault(
359         (comp_name, comp_type),
360         IfcElement(
361             uuid=uuid,
362             din276Code=din276Code,
363             refUnit=refUnit,
364             comp_name=comp_name,
365             layers=layer_list,
366         ),
367     )
368 for element in list(element_dict.values()):
369     print(element)
370
371 print("\nDONE! Ifc_extractor has returned all Elements to '__init__'!\n")
372
373 return element_dict.values()

```

## localization.py

Liest sprachabhängige Variablen ein, die in der Konfigurationsdatei gesetzt worden sind.

```

1 """
2 Reference:
3   autor: Aaron Neugebauer
4   date: 2024/05/31
5 """
6
7 from archicad2elca_comps.config import languages
8
9
10 class Singleton(object):
11     _instance = None
12
13     def __new__(class_, *args, **kwargs):
14         if not isinstance(class_._instance, class_):
15             class_._instance = object.__new__(class_, *args, **kwargs)
16         return class_._instance
17
18
19 class Lang(Singleton):
20     def __init__(self, lang: str = "GERMAN"):
21         self._local: dict[str, dict[str, str]] = languages
22         self.set_lang(lang)
23
24     def set_lang(self, lang: str):
25         self._language = lang
26         try:
27             for key, value in self._local[lang].items():
28                 self.__setattr__(key, value)
29         except KeyError:
30             raise RuntimeError(
31                 f"Language '{lang}' is not defined in 'config.py'!")
32         ) from None

```

## xml\_builder.py

Baut anhand von Objekten XML-Dateien für eLCA, die vom Python-Skripts *ifc\_extractor.py* übergeben werden.

```

1 import os
2 import xml.etree.ElementTree as ET
3 from dataclasses import dataclass
4 from importlib.resources import files
5
6
7 @dataclass
8 class ElcaElement:
9     """Class for keeping track of SubElement Values of ElementTree"""
10
11     uuid: str
12     din276Code: int
13     refUnit: str
14     name: str
15     description: str | None = ""
16     elca_eol: str | None = ""
17     elca_recycling: str | None = ""
18     elca_seperation: str | None = ""
19     elca_rW: str | None = ""
20     elca_uValue: str | None = ""
21     quantity: str | None = "1"

```

```

22
23
24 @dataclass
25 class ElcaComponent:
26     """Class for keeping track of Component Values of Components Tree"""
27
28     processConfigUuid: str
29     processConfigName: str
30     layerPosition: str
31     layerSize: str
32     lifeTime: str
33     lifeTimeDelay: str | None = "0"
34     isLayer: str | None = "true"
35     calcLca: str | None = "true"
36     isExtant: str | None = "false"
37     layerAreaRatio: str | None = "1.0"
38     layerLength: str | None = "1.0"
39     layerWidth: str | None = "1.0"
40
41
42 def save_comp(element_root, element):
43     """
44     Writes xml-strings with ElcaComponent class from given element object.
45
46     Parameters:
47         element_root:
48             Xml root where xml-strings are to be added.
49         element:
50             Object with all information to create object from ElcaComponent.
51     """
52     components_root = element_root[1]
53     for layer in element.layers:
54         ET.SubElement(components_root, "component")
55
56     c = ElcaComponent(
57         # isLayer=layer.is_layer,
58         processConfigUuid=str(layer.layer_uuid),
59         processConfigName=str(layer.layer_material),
60         lifeTime=str(layer.layer_lifeTime),
61         lifeTimeDelay=str(layer.layer_lifeTimeDelay),
62         calcLca=str(layer.calcLca).lower(),
63         isExtant=str(layer.isExtant).lower(),
64         layerPosition=str(layer.layer_position),
65         layerSize=str(layer.layer_size),
66         # layerAreaRatio=layer.layer_areaRatio,
67         # layerLength=layer.layer_length,
68         # layerWidth=layer.layer_width,
69     )
70
71     components_root[-1].set("isLayer", c.isLayer)
72     components_root[-1].set("processConfigUuid", c.processConfigUuid)
73     components_root[-1].set("processConfigName", c.processConfigName)
74     components_root[-1].set("lifeTime", c.lifeTime)
75     components_root[-1].set("lifeTimeDelay", c.lifeTimeDelay)
76     components_root[-1].set("calcLca", c.calcLca)
77     components_root[-1].set("isExtant", c.isExtant)
78     components_root[-1].set("layerPosition", c.layerPosition)
79     components_root[-1].set("layerSize", c.layerSize)
80     components_root[-1].set("layerAreaRatio", c.layerAreaRatio)
81     components_root[-1].set("layerLength", c.layerLength)
82     components_root[-1].set("layerWidth", c.layerWidth)
83
84
85 def xml_builder(exportfolder_path, elements):
86     """
87     Adds xml-strings to 'xml-template' with ElcaElement class from given element object.
88
89     Parameters:
90         exportfolder_path:
91             Destinationfolder to write finished xml-files to.

```

```

92     element:
93         Object with all information to create object from ElcaComponent.
94     """
95     ET.register_namespace("", "https://www.bauteileditor.de")
96     for element in elements:
97         xml_template = ET.parse(
98             files("archicad2elca_comps.resources").joinpath("elca_template.xml")
99         )
100    element_root = xml_template.getroot()[0]
101
102    e = ElcaElement(
103        uuid=str(element.uuid),
104        din276Code=str(element.din276Code),
105        refUnit=str(element.refUnit),
106        name=str(element.comp_name),
107        description=str(element.comp_description),
108        # elca_eol=element.elca_eol,
109        # elca_recycling=element.elca_recycling,
110        # elca_seperation=element.elca_seperation,
111        # elca_rw=element.elca_rw,
112        # elca_uValue=element.elca_uValue,
113        # quantity=element.quantity,
114    )
115
116    element_root.set("uuid", e.uuid)
117    element_root.set("din276Code", e.din276Code)
118    element_root.set("quantity", e.quantity)
119    element_root.set("refUnit", e.refUnit)
120
121    element_info_root = element_root[0]
122    element_info_root[0].text = e.name
123    element_info_root[1].text = e.description
124
125    element_attributes_root = element_root[4]
126    element_attributes_root[4][1].text = e.elca_uValue
127
128    save_comp(element_root, element)
129
130    xml_filename = f"eLCA_{e.uuid}.xml"
131    xml_template.write(
132        exportfolder_path + "/" + xml_filename,
133        encoding="UTF-8",
134        xml_declaration=True,
135        short_empty_elements=False,
136    )
137
138    print(f"{xml_filename} has been written to {exportfolder_path}")
139
140    print("\nDONE! Xml_builder has exported all Elements as eLCA readable XMLs!")

```

## elca\_template.xml

Vorlagedatei anhand das Python-Skript *xml\_builder.py* die XML-Dateien für eLCA baut.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <elca xmlns="https://www.bauteileditor.de" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.bauteileditor.de https://www.bauteileditor.de/docs/1.3/
    elca_export.xsd" schemaVersion="1.3">
3   <element>
4     <elementInfo>
5       <name><![CDATA[""]]></name>
6       <description><![CDATA[""]]></description>
7     </elementInfo>
8     <components>
9     </components>
10    <constrCatalogs>

```

```
11      <item ident="0"/>
12      <item ident="1"/>
13      <item ident="2"/>
14      <item ident="3"/>
15      <item ident="4"/>
16      <item ident="5"/>
17  </constrCatalogs>
18  <constrDesigns/>
19  <attributes>
20      <attr ident="elca.bnb.eol">
21          <caption><! [CDATA[Rueckbau]]></caption>
22      </attr>
23      <attr ident="elca.bnb.recycling">
24          <caption><! [CDATA[Verwertung]]></caption>
25      </attr>
26      <attr ident="elca.bnb.separation">
27          <caption><! [CDATA[Trennung]]></caption>
28      </attr>
29      <attr ident="elca.rW">
30          <caption><! [CDATA[R'w]]></caption>
31      </attr>
32      <attr ident="elca.uValue">
33          <caption><! [CDATA[U-Wert]]></caption>
34          <numericValue></numericValue>
35      </attr>
36  </attributes>
37  </element>
38 </elca>
```