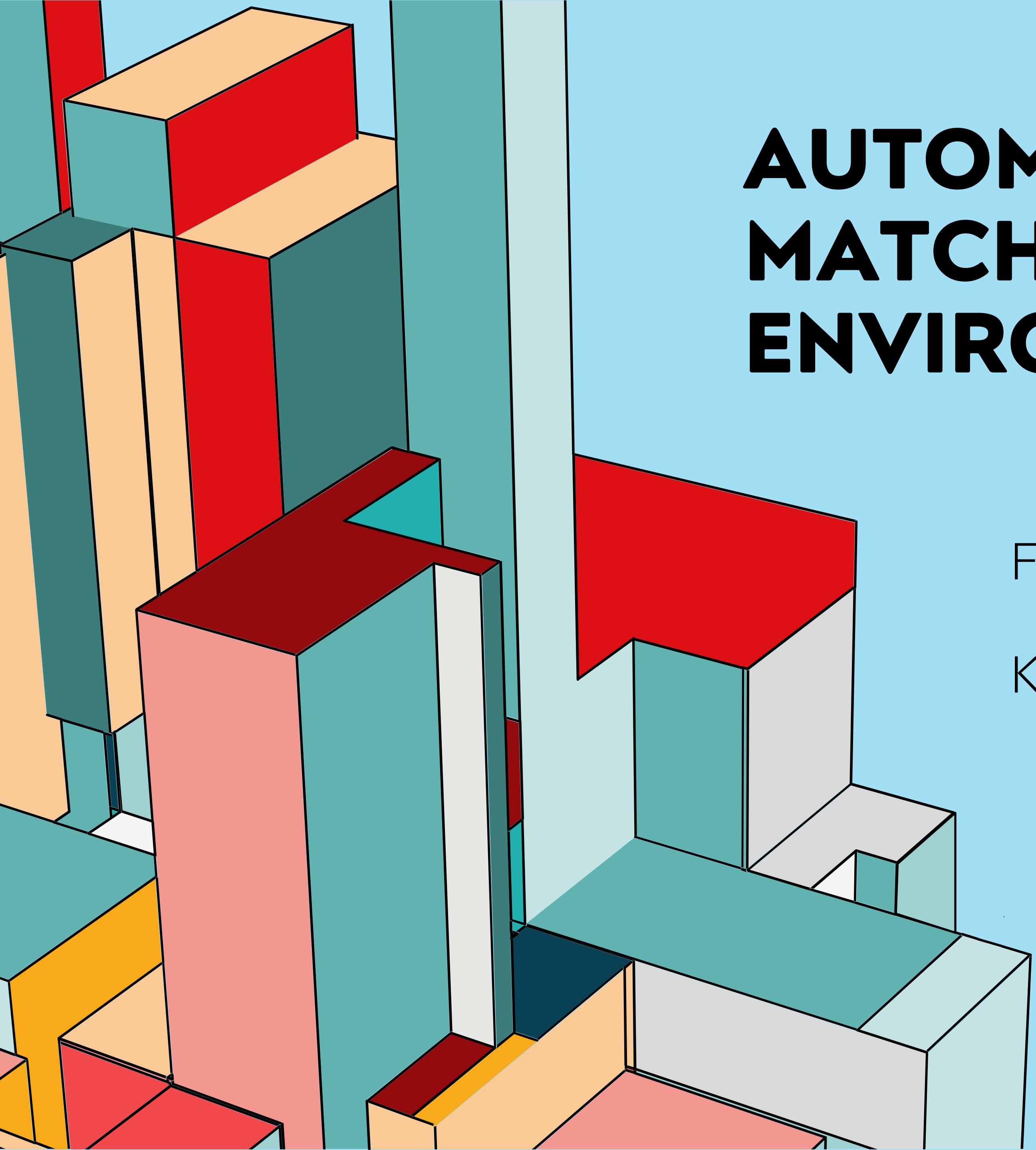


AUTOMATIZATION OF MATCHING IFC DATA AND ENVIRONMENTAL DATA

Frede Søndergaard Møllegaard s203729

Kasper Holst – s233432

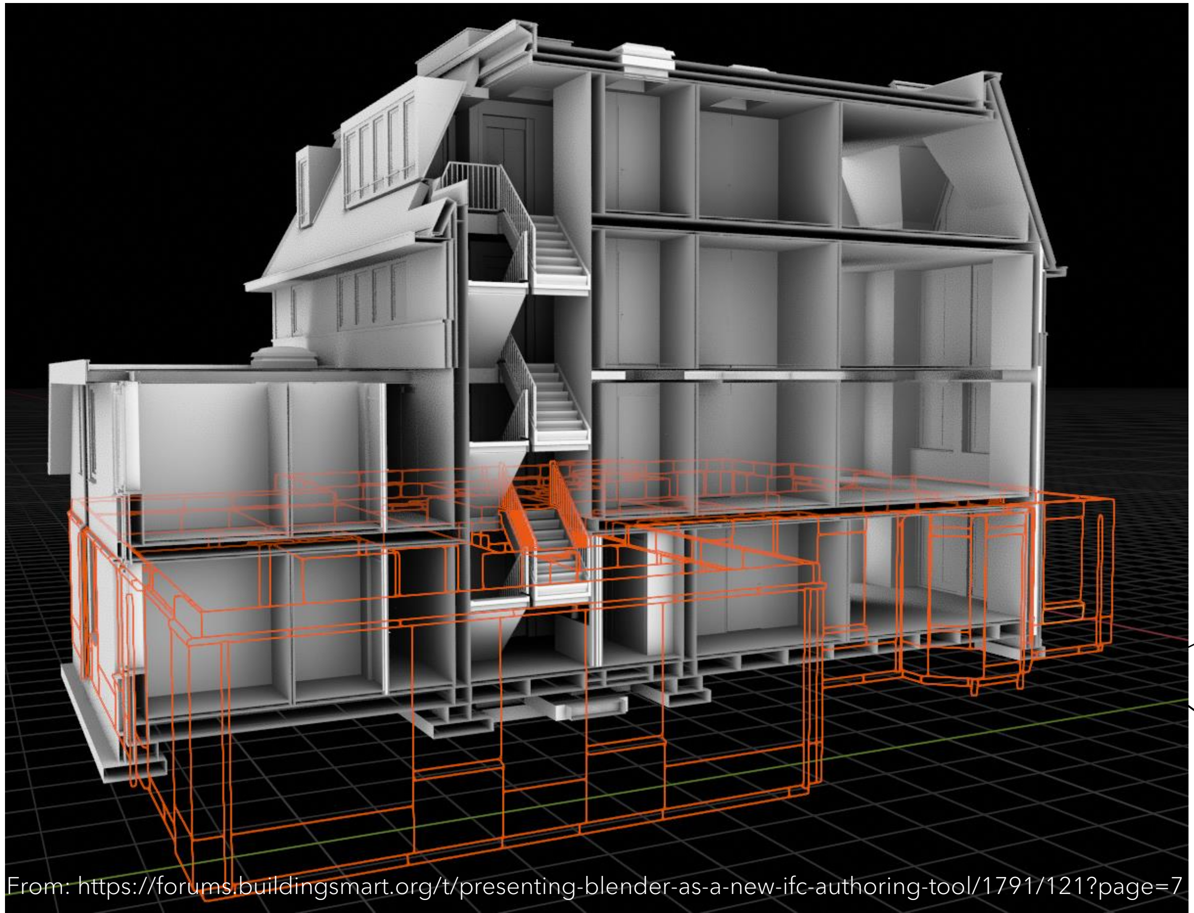
- Aim of the tool

- User of the tool

- Use case diagram

- Script of the tool

- Import Ifc data and environmental data.
- Match the materials used in the building model with the corresponding environmental data.
- Automatically generate LCI list of the building model's materials
- Reduce time used by sustainability analysts.



From: https://forums.buildingsmart.org/t/presenting-blender-as-a-new-ifc-authoring-tool/1791/121?page=7

- Aim of the tool

- User of the tool

- Use case diagram

- Script of the tool

- Early design stage: Architects, Structure and Project managers

- Late design stage: Project managers and Architects

- Propose another discipline: Sustaianability analyst
  With such a discipline we would include environmental impacts and ultimately reduce the GWP thus meeting the Building Regulations' requirements of 12 kgCO2eq/m2/year. Potentially, even meeting the Reduction Roadmaps' ambitions to reduce the impact trying to live up to the Planetary Boundaries

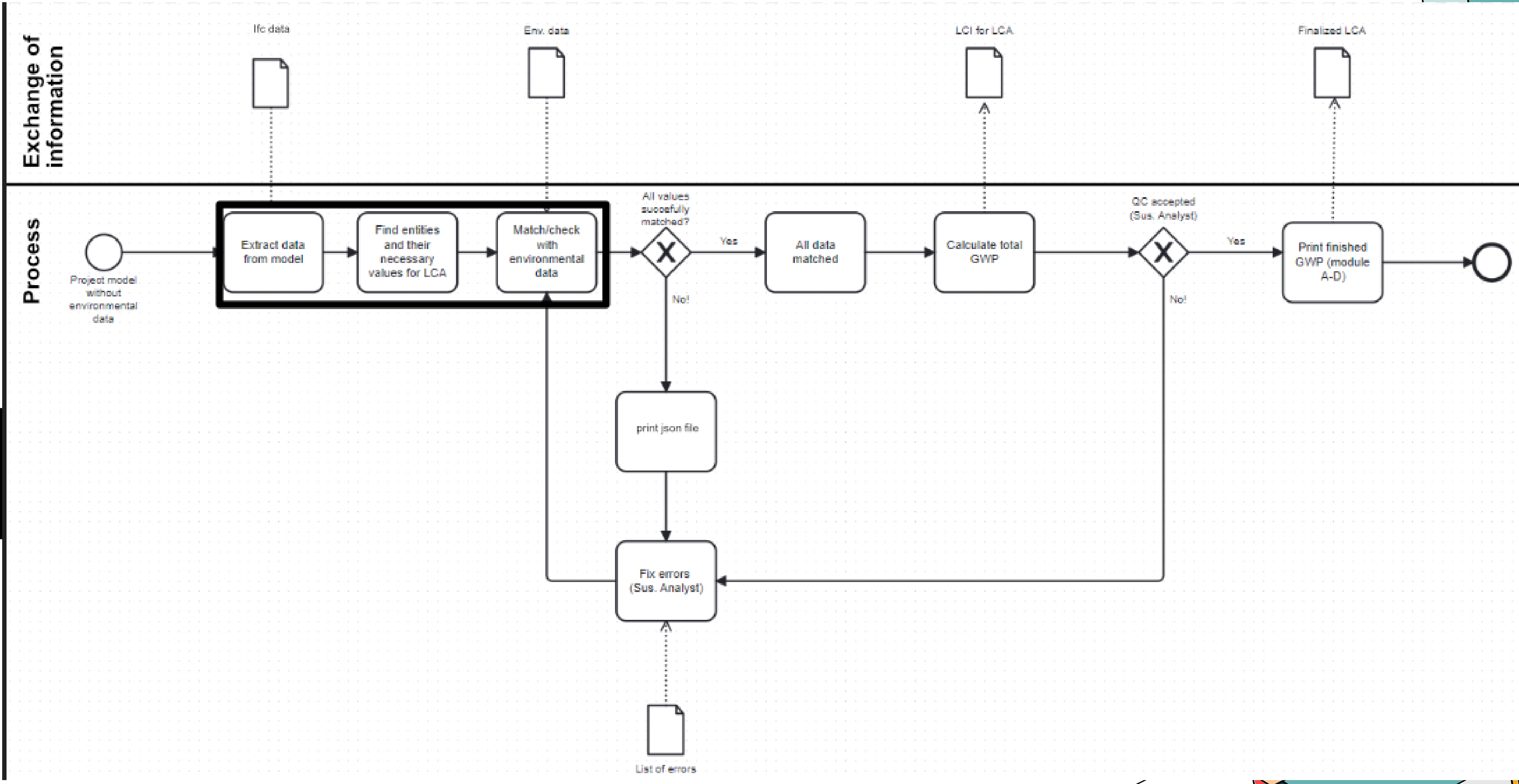
From: https://www.dbarchitect.com/about/careers

- Aim of the tool

- User of the tool

- Use case diagram

- Script of the tool

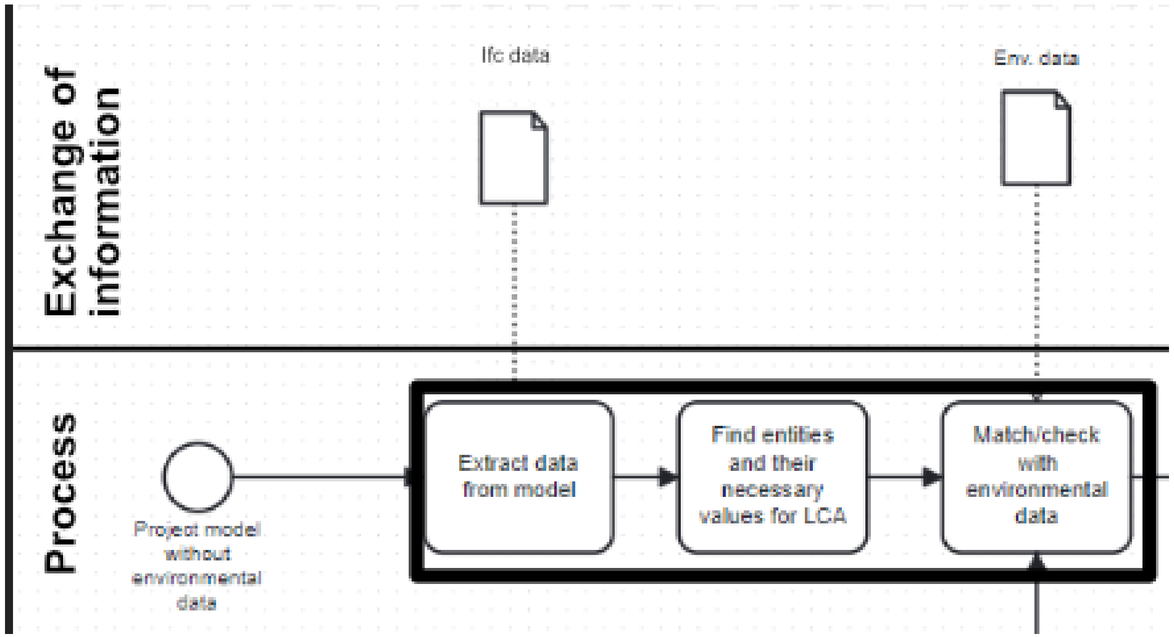

Exchange of information
Process
Ifc data
Env. data
LCI for LCA
Finalized LCA
Project model without environmental data
Extract data from model
Find entities and their necessary values for LCA
Match/check with environmental data
All values succefully matched?
Yes
No!
All data matched
Calculate total GWP
QC accepted (Sus. Analyst)
Yes
No!
Print finished GWP (module A-D)
print json file
Fix errors (Sus. Analyst)
List of errors

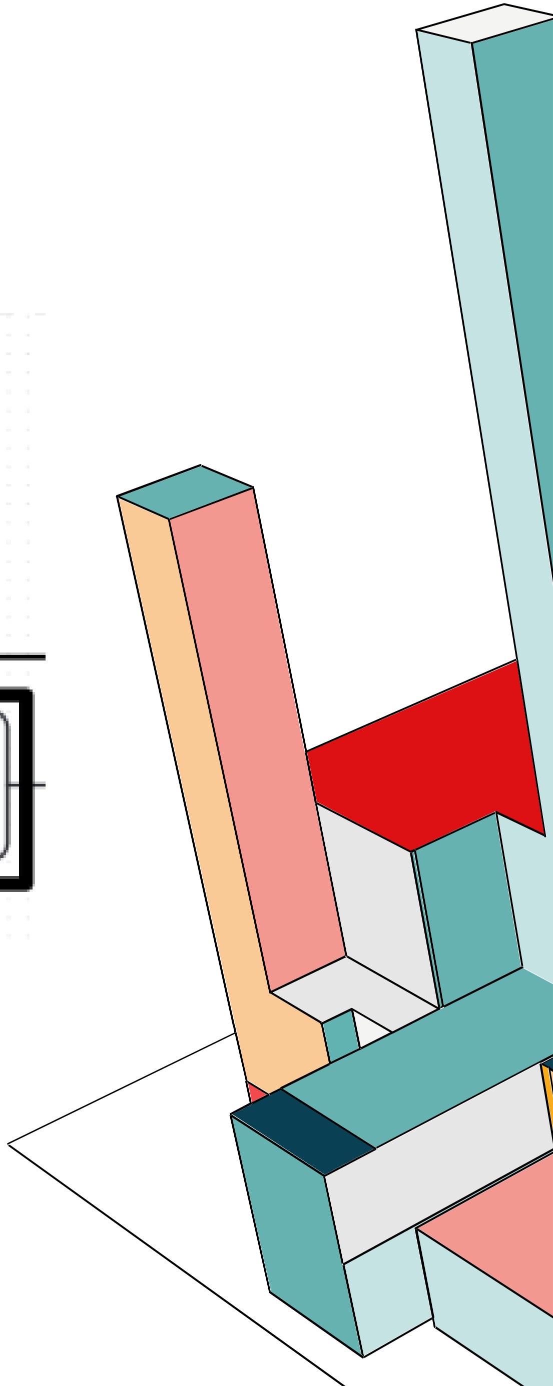
- Aim of the tool

- User of the tool

- Use case diagram

- Script of the tool


Exchange of information
Ifc data
Env. data
Process
Project model without environmental data
Extract data from model
Find entities and their necessary values for LCA
Match/check with environmental data

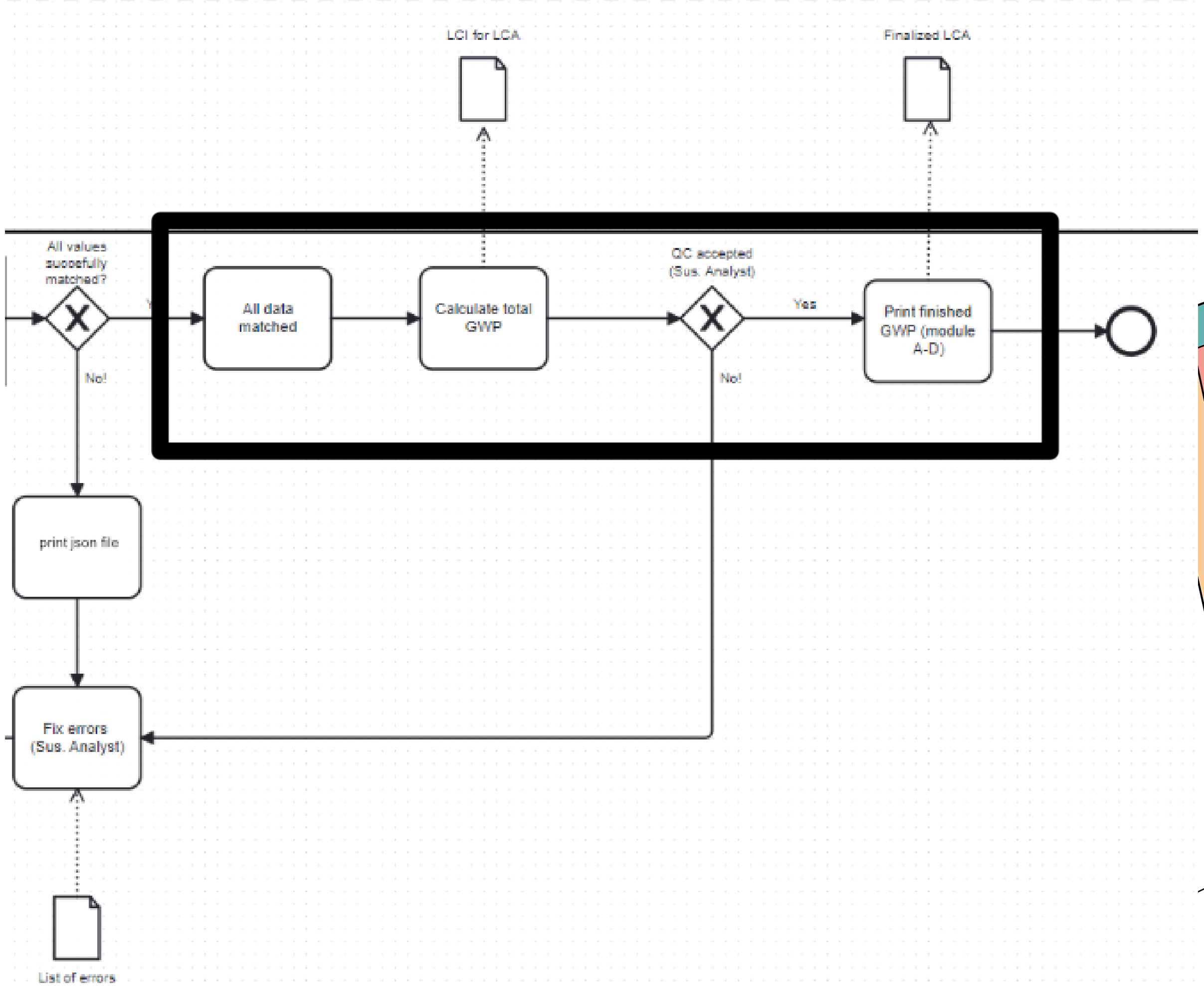


- Aim of the tool

- User of the tool

- Use case diagram

- Script of the tool



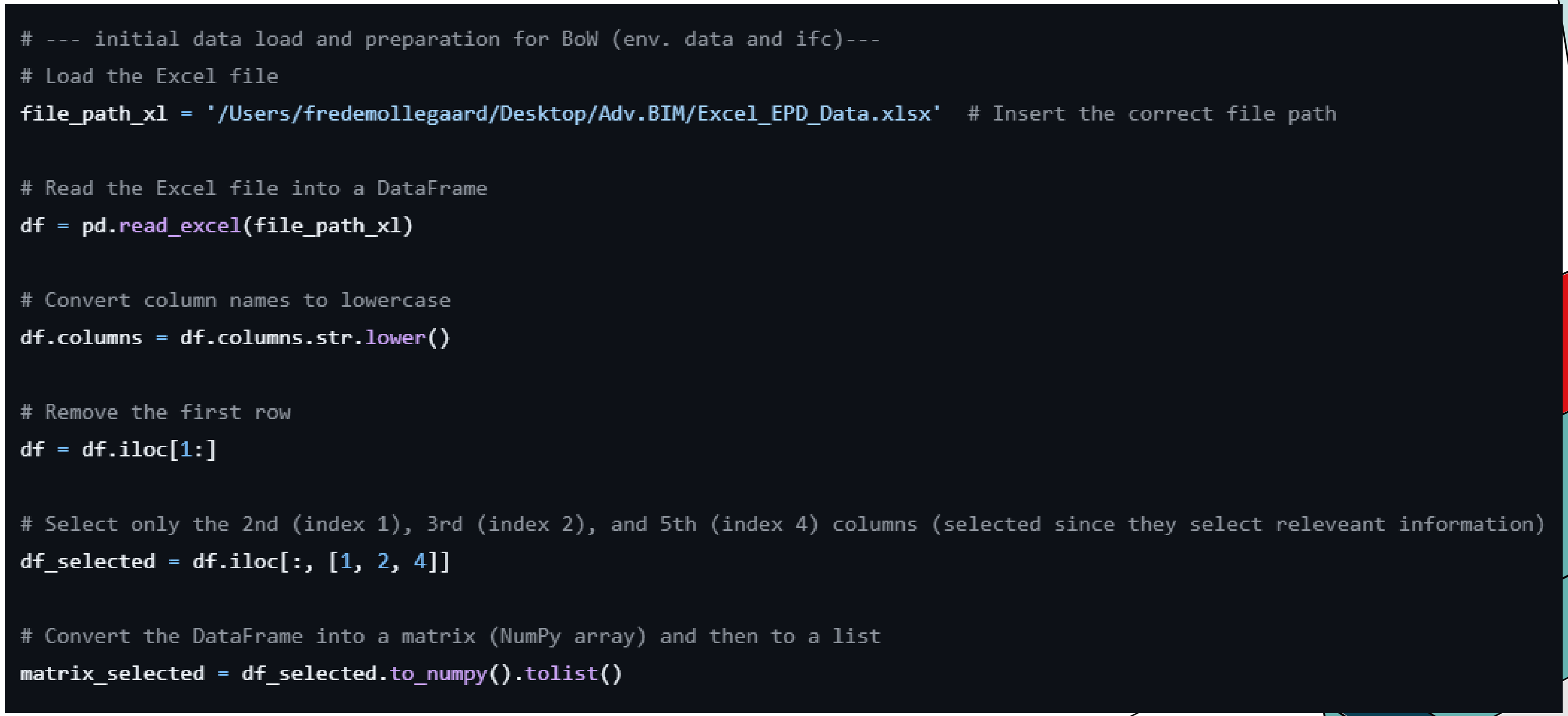
- Aim of the tool

- User of the tool

- Use case diagram

- Script of the tool

Import of environmental data and converting to cleaned-up matrix



```
# --- initial data load and preparation for BoW (env. data and ifc)---
# Load the Excel file
file_path_xl = '/Users/fredemollegaard/Desktop/Adv.BIM/Excel_EPD_Data.xlsx'  # Insert the correct file path

# Read the Excel file into a DataFrame
df = pd.read_excel(file_path_xl)

# Convert column names to lowercase
df.columns = df.columns.str.lower()

# Remove the first row
df = df.iloc[1:]

# Select only the 2nd (index 1), 3rd (index 2), and 5th (index 4) columns (selected since they select releveant information)
df_selected = df.iloc[:, [1, 2, 4]]

# Convert the DataFrame into a matrix (NumPy array) and then to a list
matrix_selected = df_selected.to_numpy().tolist()
```

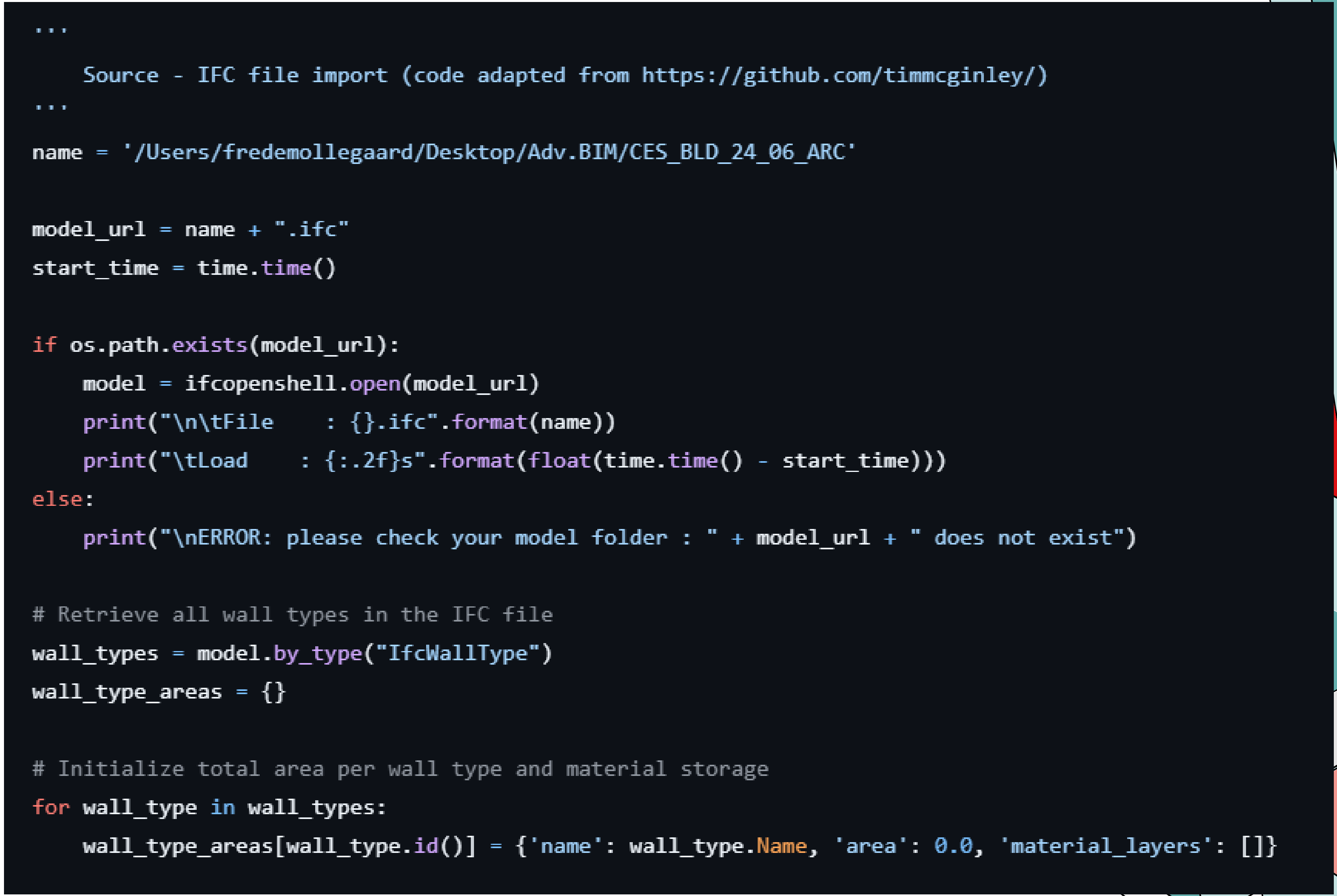
- Aim of the tool

- User of the tool

- Use case diagram

- Script of the tool

Retrieve Ifc model, print loading time and retrieve IfcEntity - IfcWall



```
'''
    Source - IFC file import (code adapted from https://github.com/timmcginley/)
'''
name = '/Users/fredemollegaard/Desktop/Adv.BIM/CES_BLD_24_06_ARC'

model_url = name + ".ifc"
start_time = time.time()

if os.path.exists(model_url):
    model = ifcopenshell.open(model_url)
    print("\n\tFile    : {}.ifc".format(name))
    print("\tLoad    : {:.2f}s".format(float(time.time() - start_time)))
else:
    print("\nERROR: please check your model folder : " + model_url + " does not exist")

# Retrieve all wall types in the IFC file
wall_types = model.by_type("IfcWallType")
wall_type_areas = {}

# Initialize total area per wall type and material storage
for wall_type in wall_types:
    wall_type_areas[wall_type.id()] = {'name': wall_type.Name, 'area': 0.0, 'material_layers': []}
```

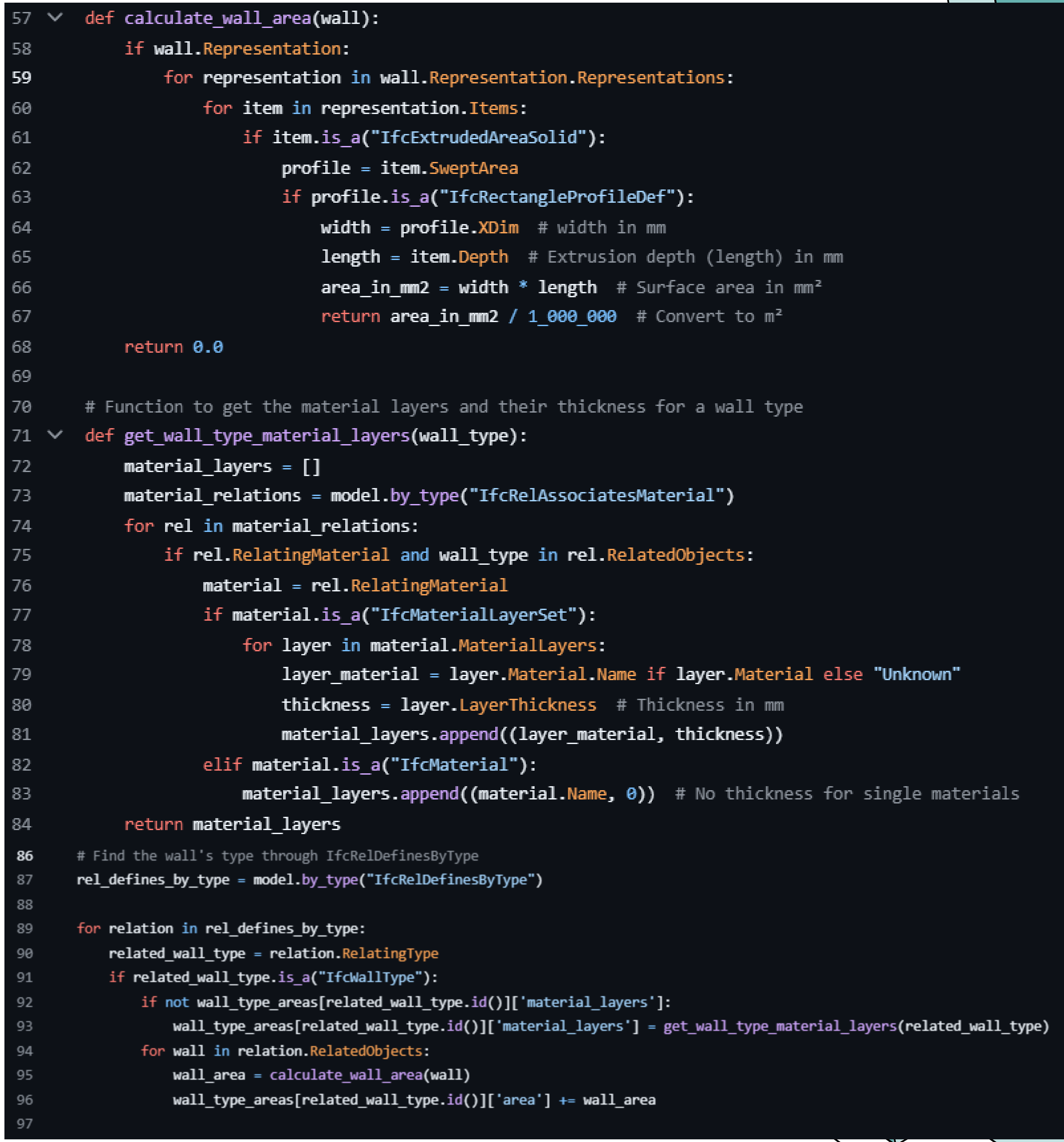
- Aim of the tool

Calculate wall area, extract material layers and thickness

- User of the tool

- Use case diagram

- Script of the tool



```python
def calculate_wall_area(wall):
    if wall.Representation:
        for representation in wall.Representation.Representations:
            for item in representation.Items:
                if item.is_a("IfcExtrudedAreaSolid"):
                    profile = item.SweptArea
                    if profile.is_a("IfcRectangleProfileDef"):
                        width = profile.XDim  # width in mm
                        length = item.Depth  # Extrusion depth (length) in mm
                        area_in_mm2 = width * length  # Surface area in mm²
                        return area_in_mm2 / 1_000_000  # Convert to m²
    return 0.0

# Function to get the material layers and their thickness for a wall type
def get_wall_type_material_layers(wall_type):
    material_layers = []
    material_relations = model.by_type("IfcRelAssociatesMaterial")
    for rel in material_relations:
        if rel.RelatingMaterial and wall_type in rel.RelatedObjects:
            material = rel.RelatingMaterial
            if material.is_a("IfcMaterialLayerSet"):
                for layer in material.MaterialLayers:
                    layer_material = layer.Material.Name if layer.Material else "Unknown"
                    thickness = layer.LayerThickness  # Thickness in mm
                    material_layers.append((layer_material, thickness))
            elif material.is_a("IfcMaterial"):
                material_layers.append((material.Name, 0))  # No thickness for single materials
    return material_layers
# Find the wall's type through IfcRelDefinesByType
rel_defines_by_type = model.by_type("IfcRelDefinesByType")

for relation in rel_defines_by_type:
    related_wall_type = relation.RelatingType
    if related_wall_type.is_a("IfcWallType"):
        if not wall_type_areas[related_wall_type.id()]['material_layers']:
            wall_type_areas[related_wall_type.id()]['material_layers'] = get_wall_type_material_layers(related_wall_type)
        for wall in relation.RelatedObjects:
            wall_area = calculate_wall_area(wall)
            wall_type_areas[related_wall_type.id()]['area'] += wall_area

```

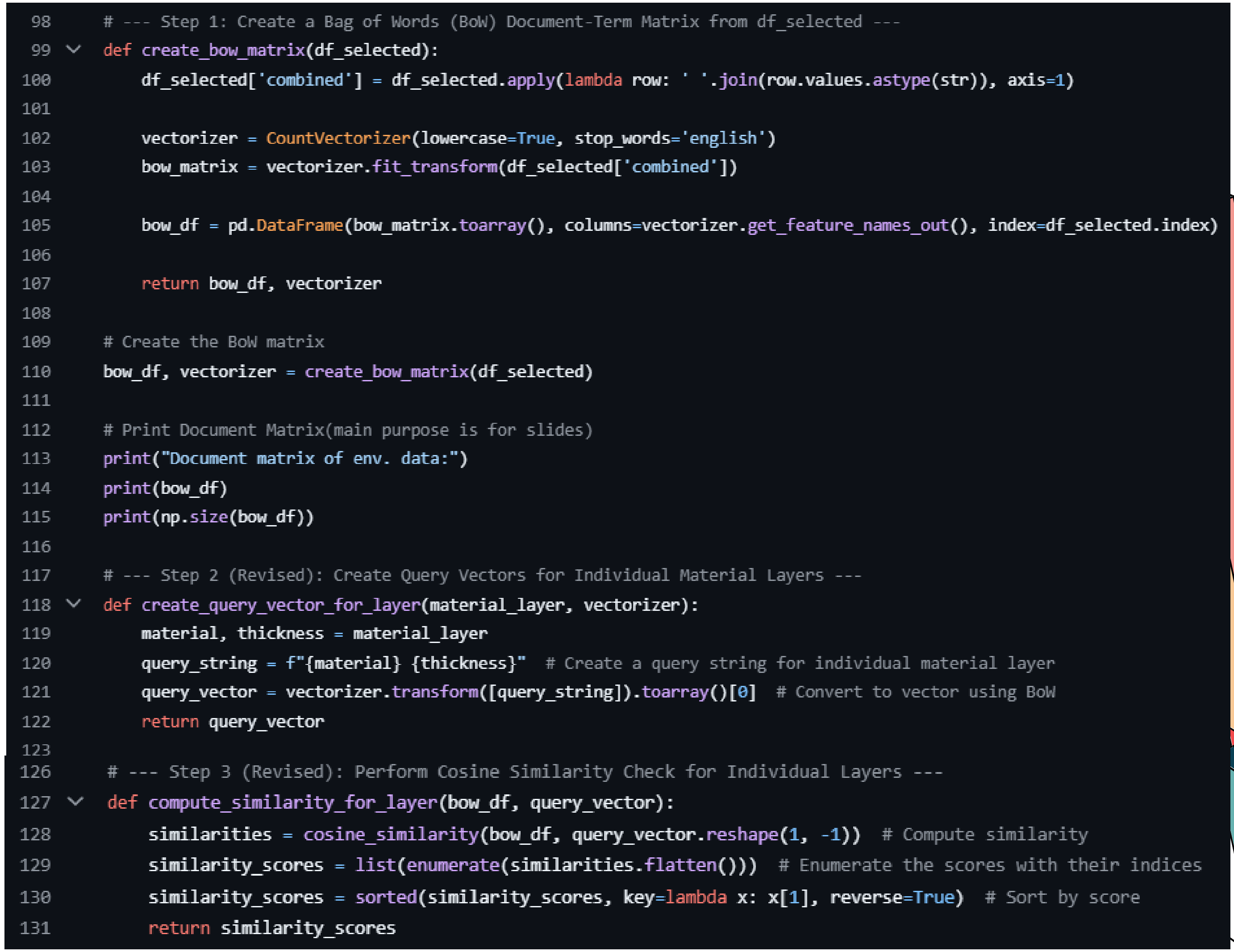
- Aim of the tool

- User of the tool

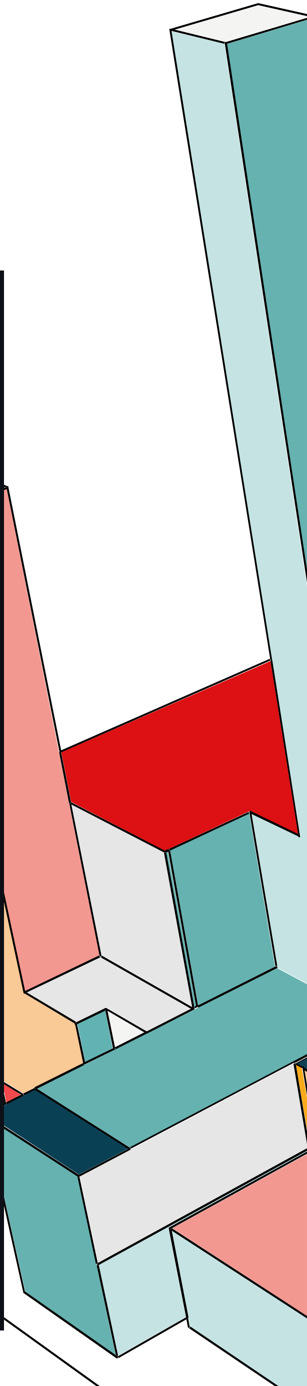
- Use case diagram

- Script of the tool

Creating a BoW matrix, creating a query vector for each material layer, and perform a cosine similarity check for individual layers matching the material layers from the Ifc file with the environmental data in the excel sheet.



```python
# --- Step 1: Create a Bag of Words (BoW) Document-Term Matrix from df_selected ---
def create_bow_matrix(df_selected):
    df_selected['combined'] = df_selected.apply(lambda row: ' '.join(row.values.astype(str)), axis=1)

    vectorizer = CountVectorizer(lowercase=True, stop_words='english')
    bow_matrix = vectorizer.fit_transform(df_selected['combined'])

    bow_df = pd.DataFrame(bow_matrix.toarray(), columns=vectorizer.get_feature_names_out(), index=df_selected.index)

    return bow_df, vectorizer

# Create the BoW matrix
bow_df, vectorizer = create_bow_matrix(df_selected)

# Print Document Matrix(main purpose is for slides)
print("Document matrix of env. data:")
print(bow_df)
print(np.size(bow_df))

# --- Step 2 (Revised): Create Query Vectors for Individual Material Layers ---
def create_query_vector_for_layer(material_layer, vectorizer):
    material, thickness = material_layer
    query_string = f"{material} {thickness}"  # Create a query string for individual material layer
    query_vector = vectorizer.transform([query_string]).toarray()[0]  # Convert to vector using BoW
    return query_vector

# --- Step 3 (Revised): Perform Cosine Similarity Check for Individual Layers ---
def compute_similarity_for_layer(bow_df, query_vector):
    similarities = cosine_similarity(bow_df, query_vector.reshape(1, -1))  # Compute similarity
    similarity_scores = list(enumerate(similarities.flatten()))  # Enumerate the scores with their indices
    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)  # Sort by score
    return similarity_scores
```



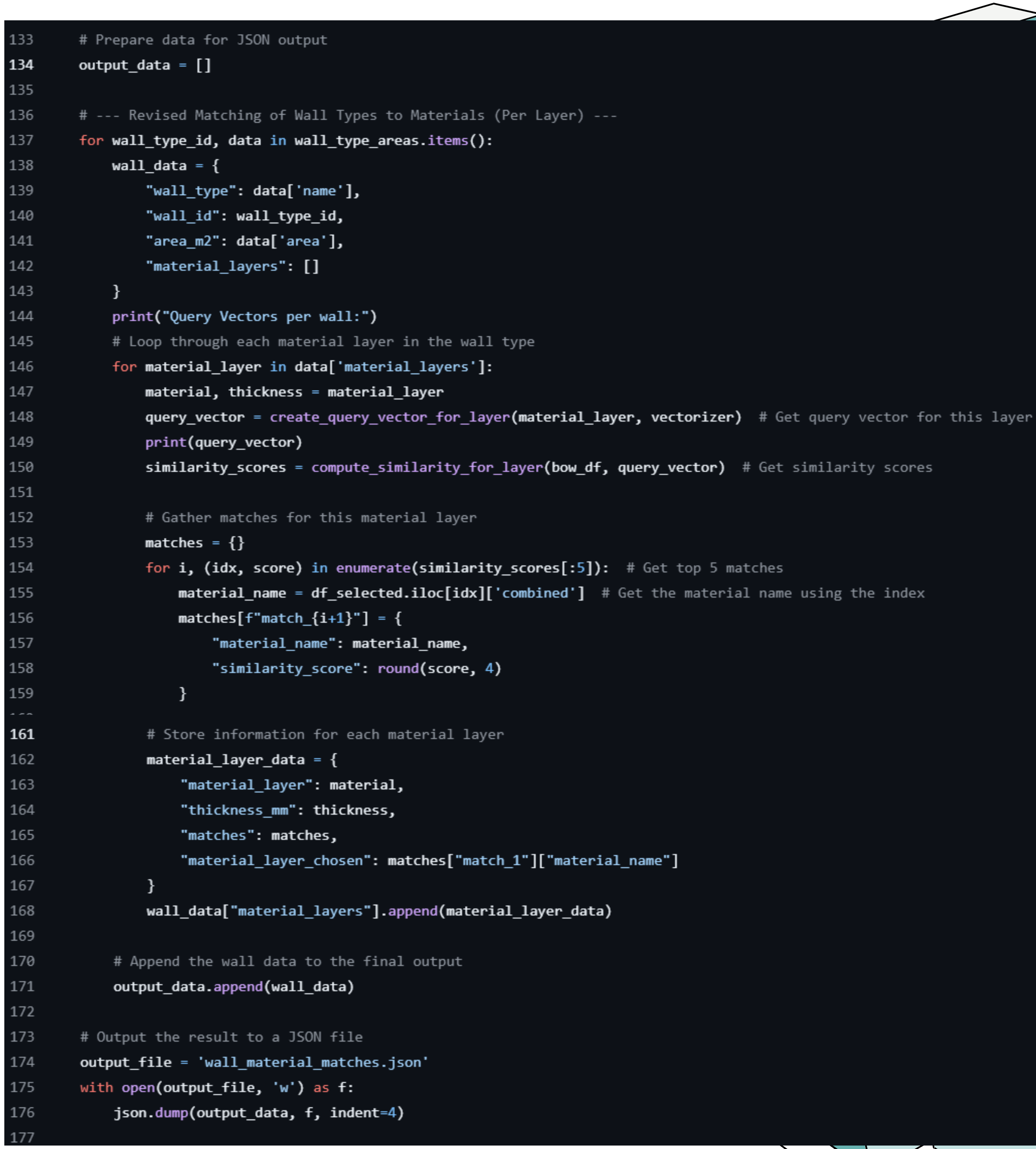
- Aim of the tool

- User of the tool

- Use case diagram

- Script of the tool

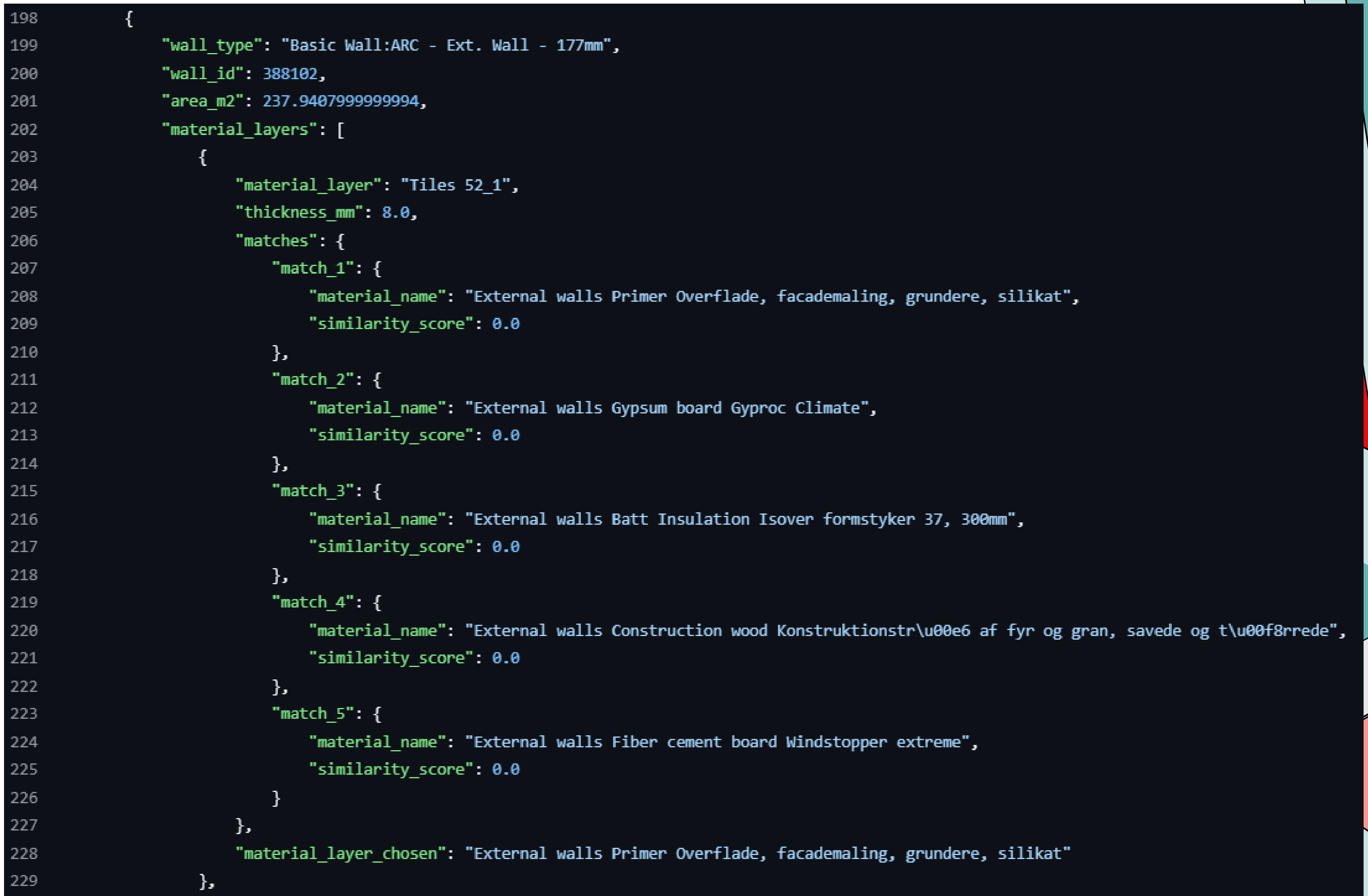
Export the data that has been found by the tool into a JSON file



```python
# Prepare data for JSON output
output_data = []

# --- Revised Matching of Wall Types to Materials (Per Layer) ---
for wall_type_id, data in wall_type_areas.items():
    wall_data = {
        "wall_type": data['name'],
        "wall_id": wall_type_id,
        "area_m2": data['area'],
        "material_layers": []
    }
    print("Query Vectors per wall:")
    # Loop through each material layer in the wall type
    for material_layer in data['material_layers']:
        material, thickness = material_layer
        query_vector = create_query_vector_for_layer(material_layer, vectorizer)  # Get query vector for this layer
        print(query_vector)
        similarity_scores = compute_similarity_for_layer(bow_df, query_vector)  # Get similarity scores

        # Gather matches for this material layer
        matches = {}
        for i, (idx, score) in enumerate(similarity_scores[:5]):  # Get top 5 matches
            material_name = df_selected.iloc[idx]['combined']  # Get the material name using the index
            matches[f"match_{i+1}"] = {
                "material_name": material_name,
                "similarity_score": round(score, 4)
            }

        # Store information for each material layer
        material_layer_data = {
            "material_layer": material,
            "thickness_mm": thickness,
            "matches": matches,
            "material_layer_chosen": matches["match_1"]["material_name"]
        }
        wall_data["material_layers"].append(material_layer_data)

    # Append the wall data to the final output
    output_data.append(wall_data)

# Output the result to a JSON file
output_file = 'wall_material_matches.json'
with open(output_file, 'w') as f:
    json.dump(output_data, f, indent=4)
```

- Aim of the tool
- User of the tool
- Use case diagram
- Script of the tool

An extraction from the JSON file, showing what IfcWall the IfcMaterialLayer is connected to, the wall id, the area of the wall, layer name of material, material thickness, the five best matches (from the environmental data) and finally the chosen material matched with.



```
        {
            "wall_type": "Basic Wall:ARC - Ext. Wall - 177mm",
            "wall_id": 388102,
            "area_m2": 237.9407999999994,
            "material_layers": [
                {
                    "material_layer": "Tiles 52_1",
                    "thickness_mm": 8.0,
                    "matches": {
                        "match_1": {
                            "material_name": "External walls Primer Overflade, facademaling, grundere, silikat",
                            "similarity_score": 0.0
                        },
                        "match_2": {
                            "material_name": "External walls Gypsum board Gyproc Climate",
                            "similarity_score": 0.0
                        },
                        "match_3": {
                            "material_name": "External walls Batt Insulation Isover formstyker 37, 300mm",
                            "similarity_score": 0.0
                        },
                        "match_4": {
                            "material_name": "External walls Construction wood Konstruktionstr\u00e6 af fyr og gran, savede og t\u00f8rrede",
                            "similarity_score": 0.0
                        },
                        "match_5": {
                            "material_name": "External walls Fiber cement board Windstopper extreme",
                            "similarity_score": 0.0
                        }
                    },
                    "material_layer_chosen": "External walls Primer Overflade, facademaling, grundere, silikat"
                },
```

THANK YOU

Frede Søndergaard Møllegaard s203729

Kasper Holst – s233432