Técnicas y Herramientas - 2021
Trabajo Final

Martin Urbieta

17 de septiembre de 2021

Resumen

El presente documento es un informe del trabajo final del curso 'Técnicas y Herramientas' dictado en el primer semestre de 2021, correspondiente a la carrera de Maestría en Ingeniería de Software dictada por la Facultad de Informática de la Universidad Nacional de La Plata.

1. Introducción

En el marco del trabajo final de la cátedra, se ha propuesto al cuerpo docente, aplicar las técnicas modernas de la programación, en particular en aspectos de la programación orientada a objetos y patrones de diseño [GHJV93], y en especial a los *Creational Patterns*, al código que forma parte del plan de trabajo de investigación del doctorado en curso, desarrollado en Python para instanciar modelos Industry Foundation Clases (IFC)[1] de estructuras de edificios. El código en cuestión tiene como responsabilidad recibir archivos de texto con resultados de etiquetados de elementos, obtenidos mediante la aplicación de algoritmos de *Machine Learning* que identifican elementos estructurales en láminas de planos de estructuras de edificio, y procesar los datos para generar un modelo digital de la estructura. Atento que el código requiere de la librería IfcOpenShell[2], compatible con Python, no ha sido posible su migración a otros lenguajes, como JavaScript. El código intervenido para generar modelos IFC se encuentra en sus primeras etapas de prototipado, procesando los primeros los resultados de la implementación aprendizaje automático. Si bien se encuentra en las primeras etapas de desarrollo, su estado actual encuadra como *spaghetti code*[3], resultando muy provechoso para las siguientes etapas aplicar los conceptos de programación orientada a objetos (POO).

1.1. Limitaciones

El código intervenido, se encuentra en una etapa de desarrollo, aunque funcional es la primer prueba piloto para reconocimiento de vigas, columnas y losas, estimándose la altura de cada piso. Para el desarrollo de la actividad, se redujo a un caso juguete con archivo de entrada modificado de manera de lograr un test de funcionalidad.

1.2. Objetivos

Los objetivos principales han sido implementar el paradigma de POO al código espagueti o *spaghetti code*. Las actividades consistieron en:

- Elaboracion de diagrama UML
- Refactoring de código implementando POO
- Test de unidad (en Python)

[1] https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/
[2] http://ifcopenshell.org/
[3] http://www.laputan.org/mud/

2. Entorno de trabajo

2.1. Sistema Operativo

El sistema utilizado ha sido Ubuntu 20.04.2 LTS, que se adaptó a las librerías y soporte disponible, mayoritariamente Open Sourse.

2.2. Entorno de desarrollo

De igual manera que el trabajo práctico desarollado en el curso, se utilizó Visual Studio Code 1.60.1

2.3. Lenguaje

El código fue desarrollado en Python 3.8, que tiene disponible la librería IfcOpenShell que permite manipular archivos IFC.

2.4. Industrial Foundation Class

Industry Foundation Classes (IFC) es un formato de código libre y neutral, que permite describir, intercambiar y compartir información de un proyecto, durante todo su ciclo de vida. IFC es un modelo de datos estándar y abierto, utilizado en la industria de la construcción. Define las características de los datos relacionados con el diseño, construcción, mantenimiento y operación de obras civiles.

El formato define la estructura de datos de manera simple y lógica, la cual organiza los datos en una estructura de árbol de acuerdo a su tipo.

Los archivos IFC facilitan la interoperatibilidad entre plataformas de software.

Para el caso de una viga IFC, las especificaciones del estándar es la indicada en la imagen 1 .

3. Desarrollo del trabajo - pipeline

3.1. Reconocimiento de imágenes

Previa recopilación de planos de construcciones, principalmente edificios, se seleccionó un primer batch de planos de estructuras, como el de la figura 2, donde indican los elementos estructurales como ser vigas, columnas, losas, sin incluirse en esta etapa otros elementos como fundaciones, escaleras, etc, tablas u otra información relevante incluida en los planos. Se procedió en una primera etapa a focalizar el reconocimiento de los elementos gráficos mencionados, mediante la aplicación de un algoritmo de aprendizaje automático o machine learning. Las imágenes debieron ser preprocesada con un etiquetado de los elementos para el entrenamiento del modelo de reconocimiento, separando un batch de prueba y otro de testing, utilizando técnicas de aumentación para incrementar el dataset de prueba. El código implementado la fase de reconocimiento, no forma parte del alcance del trabajo propuesto. En esta etapa de prototipo, se ha utilizado un repositorio público que debió ser adecuado al dominio de estudio y adecuado los settings al dataset preparado. El resultado del reconocimiento es un archivo con los etiquetados obtenidos. La imagen 3 muestra el etiquetad de los elementos, y en la figura 4 se observa tanto el etiquetado como el archivo de salida. A cada etiqueta se le asigna su respectiva probabilidad. El etiquetado responde a un sistema cartesiano referido a los píxeles de las imágenes, quedando pendiente para las siguientes etapas el correcto escalado.

3.2. Elaboración de modelo IFC

El código intervenido, tiene la responsabilidad de procesar archivos de texto .TXT, obtenidos como producto de aplicar un algoritmo Machine Learning de reconocimiento de imágenes entrenado con planos de construcción etiquetados previamente con una porción del dataset, que predice resultados en nuevos planos. El código modela los elementos identificados, y genera un archivo de salida IFC que puede ser importado en alguna plataforma que soporte el formato IFC.

EXPRESS specification:

```
ENTITY IfcBeam
   SUBTYPE OF (IfcBuildingElement);
END_ENTITY;
```

Inheritance graph

```
ENTITY IfcBeam;
   ENTITY IfcRoot;
      GlobalId                    : IfcGloballyUniqueId;
      OwnerHistory                : IfcOwnerHistory;
      Name                        : OPTIONAL IfcLabel;
      Description                 : OPTIONAL IfcText;
   ENTITY IfcObjectDefinition;
   INVERSE
      HasAssignments              : SET OF IfcRelAssigns FOR RelatedObjects;
      IsDecomposedBy              : SET OF IfcRelDecomposes FOR RelatingObject;
      Decomposes                  : SET [0:1] OF IfcRelDecomposes FOR RelatedObjects;
      HasAssociations             : SET OF IfcRelAssociates FOR RelatedObjects;
   ENTITY IfcObject;
      ObjectType                  : OPTIONAL IfcLabel;
   INVERSE
      IsDefinedBy                 : SET OF IfcRelDefines FOR RelatedObjects;
   ENTITY IfcProduct;
      ObjectPlacement             : OPTIONAL IfcObjectPlacement;
      Representation              : OPTIONAL IfcProductRepresentation;
   INVERSE
      ReferencedBy                : SET OF IfcRelAssignsToProduct FOR RelatingProduct;
   ENTITY IfcElement;
      Tag                         : OPTIONAL IfcIdentifier;
   INVERSE
      HasStructuralMember         : SET OF IfcRelConnectsStructuralElement FOR RelatingElement;
      FillsVoids                  : SET [0:1] OF IfcRelFillsElement FOR RelatedBuildingElement;
      ConnectedTo                 : SET OF IfcRelConnectsElements FOR RelatingElement;
      HasCoverings                : SET OF IfcRelCoversBldgElements FOR RelatingBuildingElement;
      HasProjections              : SET OF IfcRelProjectsElement FOR RelatingElement;
      ReferencedInStructures      : SET OF IfcRelReferencedInSpatialStructure FOR RelatedElements;
      HasPorts                    : SET OF IfcRelConnectsPortToElement FOR RelatedElement;
      HasOpenings                 : SET OF IfcRelVoidsElement FOR RelatingBuildingElement;
      IsConnectionRealization     : SET OF IfcRelConnectsWithRealizingElements FOR RealizingElements;
      ProvidesBoundaries          : SET OF IfcRelSpaceBoundary FOR RelatedBuildingElement;
      ConnectedFrom               : SET OF IfcRelConnectsElements FOR RelatedElement;
      ContainedInStructure        : SET [0:1] OF IfcRelContainedInSpatialStructure FOR RelatedElements;
   ENTITY IfcBuildingElement;
   ENTITY IfcBeam;
END_ENTITY;
```

Figura 1: IFC Standard - Especificaciones IfcBeam.

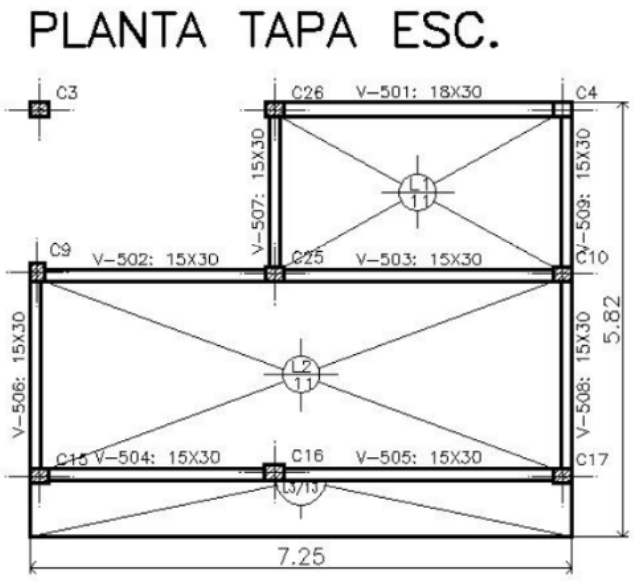

Figura 2: Ejemplo de plano estructural

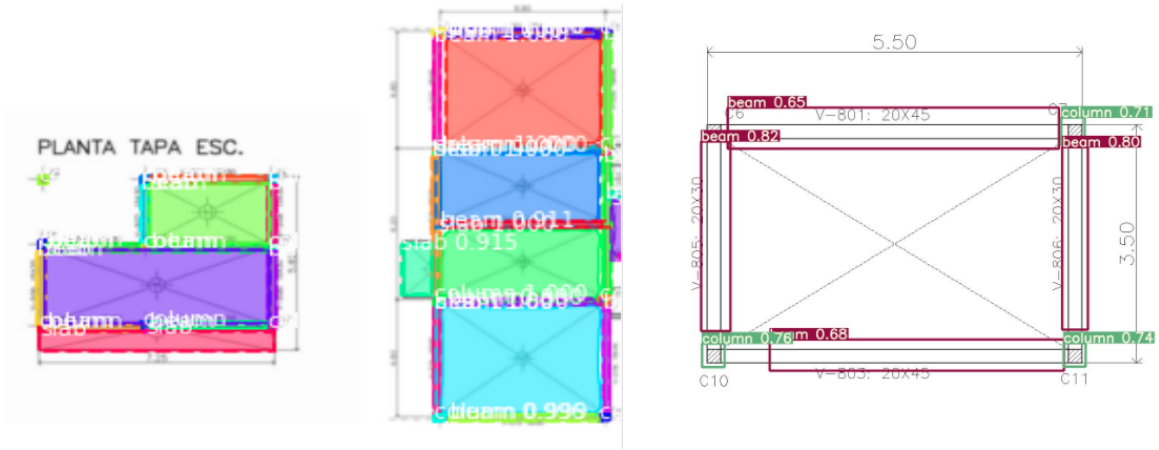

Figura 3: Reconocimiento de elementos estructurales en planos

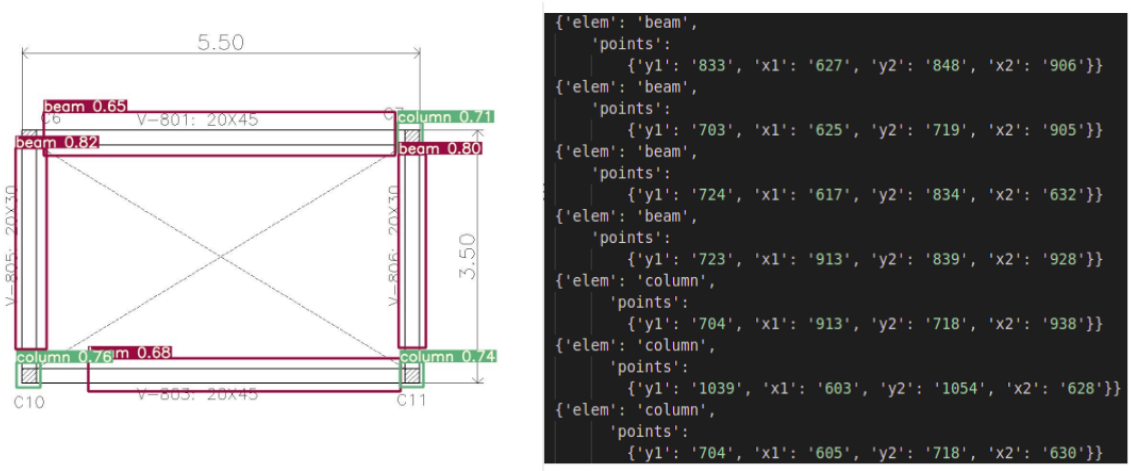

Figura 4: Etiquetado de imagen y archivo de salida

3.3. Código Original

A continuación se transcribe el código original a intervenir, producto de refactoring de scripts de ejemplos obtenido de los sitios relacionados a la librería IfcOpenShell, para poder procesar las entradas JSON y modelar los elementos objetivos.

El código puede ser consultado en la sección **??**

3.4. Archivo de Ingreso

Se han utilizado dos archivos de ingreso juguete, uno con las predicciones del algoritmo machine learning y otro que indica las relaciones entre los elementos. Este ultimo es resultado de aplicar un algoritmo de distancias minimas a etiquetados, con el objetivo de reflejar relaciones estructurales entre los elementos a modelar, como por ejemplo, en la estructura de la imagen 2 la viga V-505 apoya en las columnas C16 y C17.

```
{'elem': 'column', 'tag': 'C1', 'points': {'y1': '0', 'x1': '0', 'y2': '2', 'x2': '2'
    }}
{'elem': 'column', 'tag': 'C2', 'points': {'y1': '40', 'x1': '0', 'y2': '42', 'x2': '2
    '}}
{'elem': 'column', 'tag': 'C3', 'points': {'y1': '40', 'x1': '40', 'y2': '42', 'x2': '
    42'}}
{'elem': 'column', 'tag': 'C4', 'points': {'y1': '0', 'x1': '40', 'y2': '2', 'x2': '42
    '}}
{'elem': 'beam', 'tag': 'B1', 'points': {'y1': '0', 'x1': '0', 'y2': '2', 'x2': '42'}}
```

```
{'elem': 'beam', 'tag': 'B2', 'points': {'y1': '0', 'x1': '40', 'y2': '42', 'x2': '42'
    }}
{'elem': 'beam', 'tag': 'B3', 'points': {'y1': '40', 'x1': '0', 'y2': '42', 'x2': '42'
    }}
{'elem': 'beam', 'tag': 'B4', 'points': {'y1': '0', 'x1': '0', 'y2': '42', 'x2': '2'}}
{'elem': 'slab', 'tag': 'S59', 'points': {'y1': '0', 'x1': '0', 'y2': '42', 'x2': '42'
    }}
```

Listing 1: Archivo Predicciones

```
{'elem': 'beam0', 'tag': 'B1', 'points': {'y1': '0', 'x1': '0', 'y2': '2', 'x2': '42'
    }, 'relItems': [{'elem': 'column0', 'tag': 'C1', 'points': {'y1': '0', 'x1': '0', '
    y2': '2', 'x2': '2'}}, {'elem': 'column3', 'tag': 'C4', 'points': {'y1': '0', 'x1':
     '40', 'y2': '2', 'x2': '42'}}]}
{'elem': 'beam1', 'tag': 'B2', 'points': {'y1': '0', 'x1': '40', 'y2': '42', 'x2': '42
    '}, 'relItems': [{'elem': 'column2', 'tag': 'C3', 'points': {'y1': '40', 'x1': '40'
    , 'y2': '42', 'x2': '42'}}, {'elem': 'column3', 'tag': 'C4', 'points': {'y1': '0',
    'x1': '40', 'y2': '2', 'x2': '42'}}]}
{'elem': 'beam2', 'tag': 'B3', 'points': {'y1': '40', 'x1': '0', 'y2': '42', 'x2': '42
    '}, 'relItems': [{'elem': 'column1', 'tag': 'C2', 'points': {'y1': '40', 'x1': '0',
     'y2': '42', 'x2': '2'}}, {'elem': 'column2', 'tag': 'C3', 'points': {'y1': '40', '
    x1': '40', 'y2': '42', 'x2': '42'}}]}
{'elem': 'beam3', 'tag': 'B4', 'points': {'y1': '0', 'x1': '0', 'y2': '42', 'x2': '2'
    }, 'relItems': [{'elem': 'column0', 'tag': 'C1', 'points': {'y1': '0', 'x1': '0', '
    y2': '2', 'x2': '2'}}, {'elem': 'column1', 'tag': 'C2', 'points': {'y1': '40', 'x1'
    : '0', 'y2': '42', 'x2': '2'}}]}
```

Listing 2: Archivo Relacionados

3.5. Diagrama UML

A efectos de refactorizar el código, se elaboró un diagrama UML que fue evolucionando a su versión final que se presenta en la imagen 5 donde se describen las relaciones entre las diferentes clases, sus atributos y métodos. Èste diagrama facilitará la tarea de mantenimiento y organización del crecimiento del código. La primer clase en intervenir es la clase Parser, tiene la responsabilidad de convertir el JSON del archivo TXT en arreglos donde se agrupan cada elemento y se han calculado algunos parametros requeridos para su modelado, obtenidos de realizar cálculos de distancias Euclideas de las coordenadas de los boxingLabels, como ser, anchos, largos, etc. La super clase BuildingElement incluye a las clases Column (columna), Slab (losa), Beam (viga), Storey (planta) que se instancian durante la generación del modelo. Atento a que existen diferentes tipos de columnas, vigas y otros elementos como escaleras, muros, etc.., esos heredaran de esta super clase BuildingElement.

La clase IfcModelGenerator tiene la responsabilidad de procesar el parser y generar el modelo, conociendo a los elementos que deben ser modelados.

3.6. Código desarrollado

El código final esta disponible en la sección **??**

3.7. Testing

Se procedió a realizar un test de unidad funcional, que permita comprobar el tipo y cantidad de elementos modelados, que responden a los archivos de ingreso.

```
#python -m unittest bim_builder_test.py
import faulthandler; faulthandler.enable()
from IfcModelGenerator import *
import unittest

parser = Parser('predicciones_full','relacionados_full')
parser.build()
model = IfcModelGenerator(parser)
model.builder()

class TestBimBuilder_Integral(unittest.TestCase):
    def test_column(self):
```

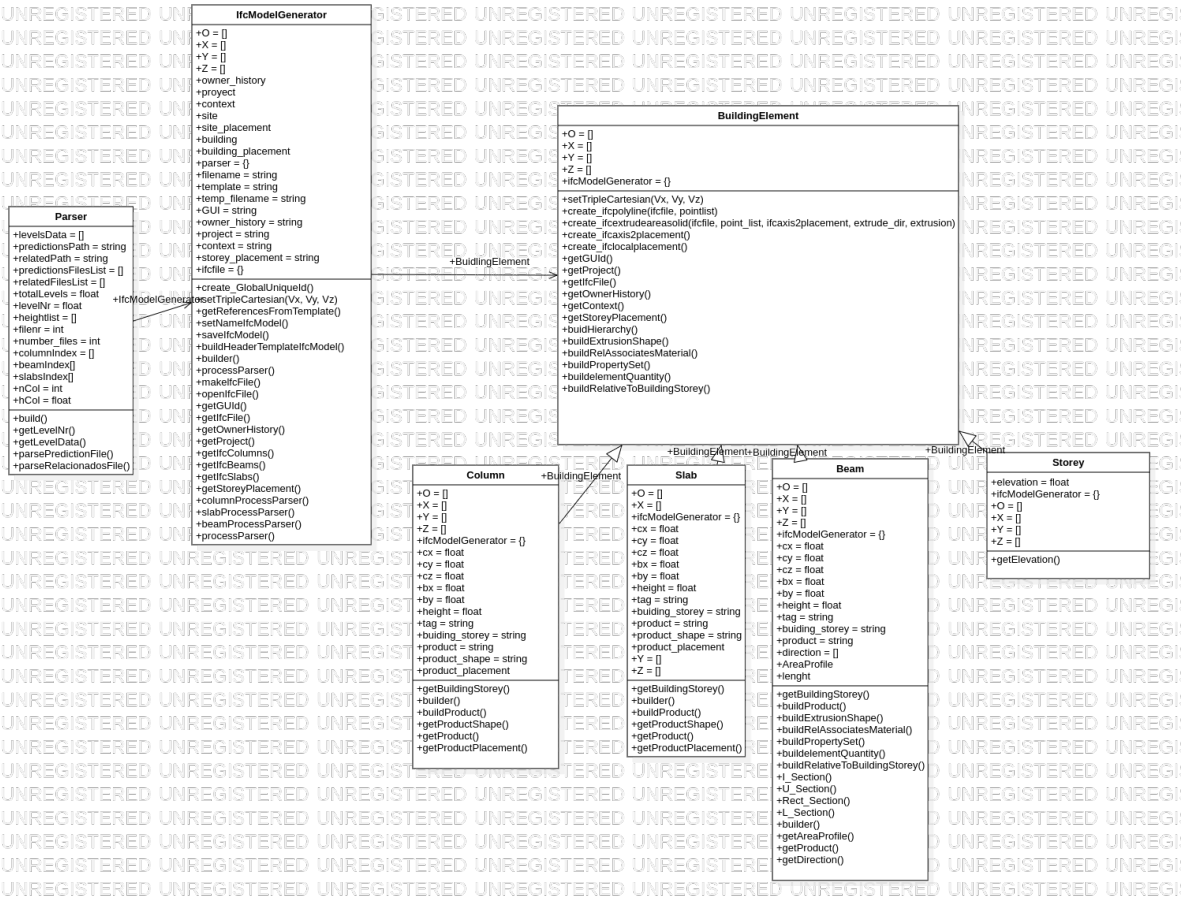

Figura 5: Diagrama UML

```
13          self.assertAlmostEqual(len(model.getIfcColumns()),4)
14      def test_beams(self):
15          self.assertAlmostEqual(len(model.getIfcBeams()),4)
16      def test_slabs(self):
17          self.assertAlmostEqual(len(model.getIfcSlabs()),1)
```

Listing 3: Test de Unidad

3.8. Resultados y mejoras para futuras etapa de desarrollo.

El test de unidad indicado en la sección 3.7 confirmo el correcto funcionamiento del código.La figura 6 es una imagen del modelo generado, importada al software Blender. Contar con un modelado automático a partir de planos existentes, reduce la necesidad de disponer recursos humanos especializados para modelar, en este caso, la estructura o bien en una siguiente etapa del desarrollo de la investigación, su arquitectura. Actualmente, sobre éstos modelos resulta posible aplicar metodologías BIM para mantenimiento, planificación de modificaciones, operación, estudios avanzados energéticos o cualquier otra en el ciclo de vida de la construcción. Si bien aun resta enriquecer el modelo con la asignación de materiales, cerramientos arquitectónicos, información semántica, etc, el modelado a través de objetos permite la manipulación de los mismos. Disponer modelos automáticos a bajo costo podrían permitir la planificación y ejecución de estudios de Data Science sobre un dominio de edificios de una ciudad.

En la sección **??** puede observarse la estructura del estándar IFC para el modelo resultante.

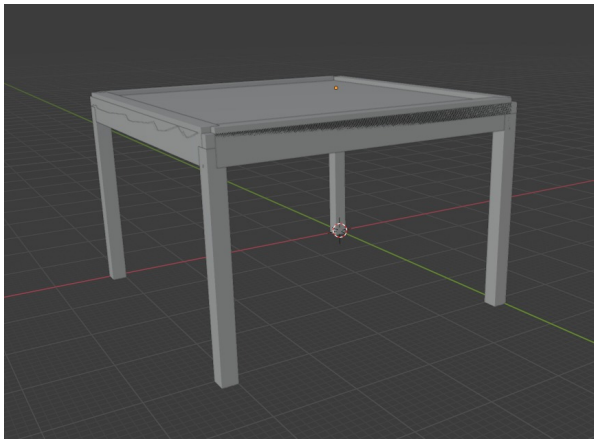

Figura 6: Modelo Estructura - Estandar IFC

3.9. Conclusiones

La implementación del paradigma Object-Oriented ha permitido facilitar el mantenimiento del código, separar las etapas de parseado y modelado. Los elementos a reconocer en planos supera a los reconocidos en la prueba piloto. La aplicación de POO brinda mayor flexibilidad para poder incluir nuevos elementos a modelar: muros, ventanas, escaleras, cubiertas, cubiertas, etc.., como así también se requiere asignar materiales, etiquetados, artefactos, muebles, etc. Por otro lado, permite mayor flexibilidad al realizar cambios en los archivos de entrada que responden a formato JSON, y que se espera enriquecer o adicionar nuevos archivos de entrada que permitan enriquecer el modelo. La implementación de test de unidad facilitaran las futuras pruebas. La adecuación del código a POO brinda mayor coherencia para la instanciación de elementos que conformaran el modelo IFC por intermedio de la librería IfcOpenShell, cuyos métodos manipulan los objetos que conforman los modelos dentro del paradigma orientado a objetos.

4. Repositorio

Repositorio GitHub[4],

Referencias

[GHJV93] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *European Conference on Object-Oriented Programming*, pages 406–431. Springer, 1993.

[4]http://https://github.com/martinurbieta/TyH2021