Semester-Long Internship Report

On

Testing Framework for Osdag OSI Files

Submitted by

Lakshana Shree S

3rd Year B.E Student, Department of CSE

Rajalakshmi Institute of Technology

Chennai

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

June 27, 2025

Acknowledgments

- I express my sincere gratitude to all those who supported and guided me throughout the course of this project. It has been a valuable and enriching experience, and I'm thankful to all who played a part in it.

- I'm especially thankful to the project staff at the Osdag team — Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia — for their consistent help, technical guidance, and kind support during my time with the project.

- I would also like to thank Prof. Siddhartha Ghosh, Principal Investigator of the Osdag project, Department of Civil Engineering, IIT Bombay, for leading the initiative and enabling students like me to be part of such an impactful project.

- My thanks to Prof. Kannan M. Moudgalya, Principal Investigator of the FOSSEE project, Department of Chemical Engineering, IIT Bombay, for making platforms like FOSSEE possible that open up learning opportunities for students across the country.

- I sincerely appreciate the efforts of Ms. Usha Viswanathan, Ms. Vineeta Parmar, and the entire FOSSEE team for their smooth coordination and support throughout the internship.

- I gratefully acknowledge the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their role in supporting and facilitating this project.

- I would also like to thank my fellow interns and teammates for their collaboration and support during this journey — working with them made the experience more engaging and enjoyable.

- Lastly, I am grateful to my college, the Department of Computer Science and Engineering, my faculty members, Head of the Department, and Principal for their constant encouragement and support throughout my studies.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock**: Collects and validates user inputs.
- **Output Dock**: Displays design results after validation.
- **CAD Window**: Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log**: Shows errors, warnings, and suggestions based on design checks.

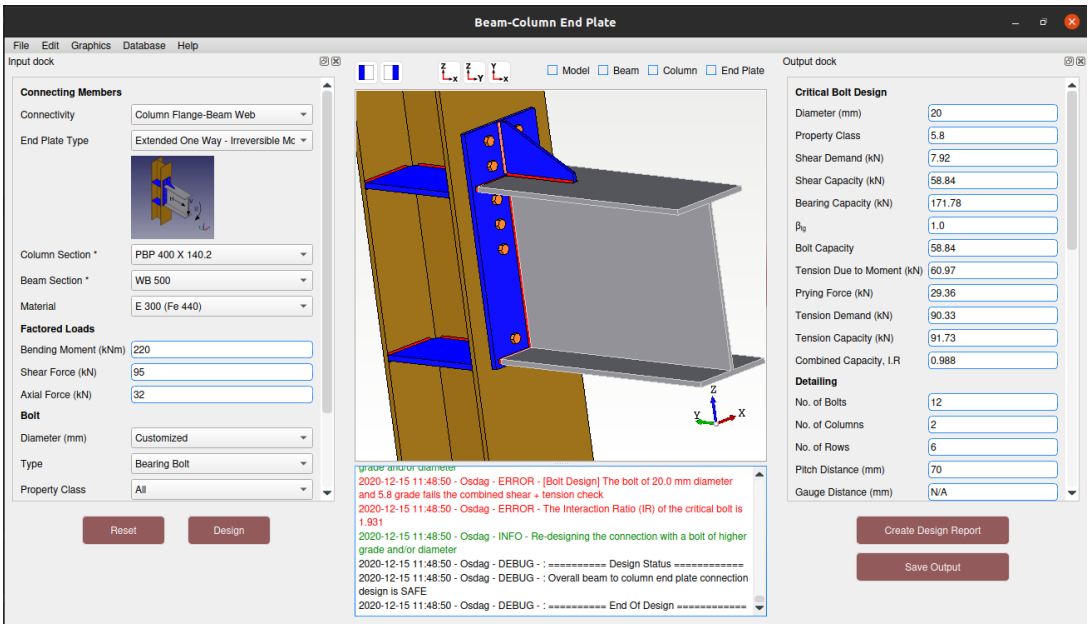

Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model**: The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences**: Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report**: Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

The goal of this project is to design a welded lap joint connecting two plates with specific thicknesses and widths, subjected to a known tensile force. The design should comply with IS 800:2007, focusing on selecting an appropriate weld size, weld material grade, and plate material grade, while ensuring that the connection is both efficient and safe.

The design problem is defined with the following parameters:

- **Plate Width** (w): Width of the plates in mm.
- **Plate Thicknesses** (t_1, t_2): Thickness of the two plates in mm.
- **Tensile Force** (P): Applied tensile force in kN.

2.1.1 Objective

Develop an algorithm to design a welded lap joint that meets the following requirements:

- Select weld size and material grade from IS 800:2007.
- Choose a steel plate grade GP from the list: {"E250", "E275", "E300", "E350", "E410"}.
- Find the most efficient connection, ensuring minimal weld length and size.
- Ensure the utilization ratio is close to 1.

- Minimize the length of the connection while maintaining safety and efficiency.
- Ensure all detailing distances are in round figures.
- Ensure the strength of the connection exceeds the tensile strength of the plate.
- The design must comply with IS 800:2007 standards.

2.2 Methodology

The design process involves the following steps:

1. **Input Parameters**: Receive values for tensile force, plate width, and thicknesses.
2. **Material Selection**: Choose appropriate weld and plate grades and determine their mechanical properties.
3. **Weld Strength Calculation**: Calculate the ultimate tensile strength and yield strength of the weld material.
4. **Design Calculation**: Determine the required weld size and the number of welds.
5. **Check Compliance**: Ensure that the design meets IS 800:2007 standards, and the utilization ratio is close to 1.
6. **Optimize Design**: Select the weld size and material grade that minimize the weld length while ensuring safety and efficiency.

2.3 Design Calculations

The calculations are performed using the following equations and considerations:

2.3.1 Input Parameters

- Plate thickness t_1, t_2 (in mm)
- Plate width w (in mm)
- Tensile force P (in kN)

- Weld size and material grade (from IS 800:2007)
- Plate grade GP (from available grades: "E250", "E275", "E300", "E350", "E410")

2.3.2 Weld Strength

The strength of the weld is calculated based on the chosen material grade. For a given material grade, the ultimate tensile strength (f_u) and yield strength (f_y) are calculated as follows:

$$f_u = 100 \times \text{Grade}$$

$$f_y = (\text{Grade} - \text{int}(\text{Grade})) \times f_u$$

2.3.3 Shear Strength of the Weld

The shear strength of the weld is calculated using the formula:

$$V_w = \text{Shear Capacity}(f_y, A_w, A_w, 0, 0, \text{Field})$$

2.3.4 Number of Welds

The required number of welds is determined by:

$$N_w = \lceil \frac{P}{V_w \times 0.75} \rceil$$

2.3.5 Design Distances

The detailing distances, such as pitch, gauge, end distance, and edge distance, are calculated based on IS 800:2007 standards.

2.3.6 Length of the Connection

The length of the connection should be minimized while ensuring that the weld strength and detailing requirements are met.

2.4 Expected Outcomes

The final design should include:

- **Weld Size** (d)
- **Weld Material Grade** (GB)
- **Length of Weld**
- **Strength of Connection**
- **Yield Strength of Plates 1 and 2** (f_y)
- **Length of Connection**
- **Efficiency of Connection** (Utilization Ratio)

2.5 Flowchart

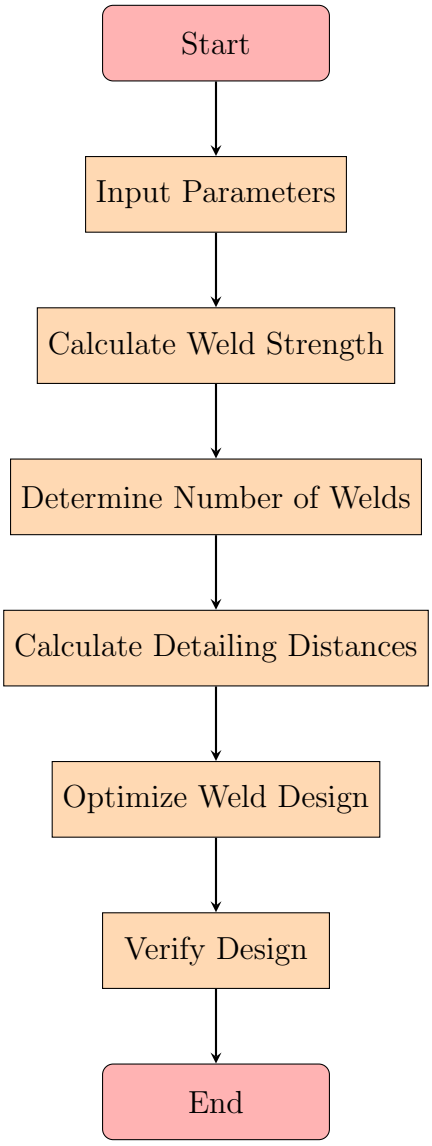

Figure 2.1: Flowchart of the Design Procedure

2.5.1 Full code

```
import math


# Define material grade properties as per IS800:2007 and IS2062
    :2011
grade_yield_strengths = {
    "E250": 250,  # MPa
    "E275": 275,
```

```
    "E300": 300,
    "E350": 350,
    "E410": 410
}

# Define design constants
partial_safety_factor = 1.25  # for weld material as per IS
    800:2007

# Function to calculate weld strength
def calculate_weld_strength(size, length, material_grade):
    # Weld shear strength (as per IS 800:2007)
    weld_strength = (grade_yield_strengths[material_grade] / math
        .sqrt(3)) / partial_safety_factor
    return weld_strength * size * length

# Function to design a lap joint
def design_lap_joint(P, w, t1, t2):
    """
    P: Tensile force in kN
    w: Plate width in mm
    t1, t2: Plate thicknesses in mm
    """
    # Convert tensile force to N (1 kN = 1000 N)
    P = P * 1000

    # Yield strengths of plates
    plate1_yield = grade_yield_strengths["E250"] * t1 * w  #
        Example plate grade: E250
    plate2_yield = grade_yield_strengths["E250"] * t2 * w  #
        Example plate grade: E250

    # Weld size selection (as per IS 800:2007, t_min = t1 or t2)
```

```
    weld_size = min(t1, t2) - 1  # Assuming a 1 mm reduction for
        throat thickness

    # Length of weld required
    material_grade = "E250"  # Example weld material grade
    weld_strength = calculate_weld_strength(weld_size, 1,
        material_grade)  # Per mm
    length_of_weld = P / weld_strength

    # Refined logic: only round length if it's a reasonable
        amount
    if length_of_weld > 100:  # Arbitrary threshold to prevent
        excessive rounding
        length_of_weld = math.ceil(length_of_weld / 10) * 10
    else:
        length_of_weld = round(length_of_weld, 2)

    # Connection efficiency (utilization ratio)
    connection_strength = calculate_weld_strength(weld_size,
        length_of_weld, material_grade)
    efficiency = connection_strength / max(plate1_yield,
        plate2_yield)

    # Output results
    return {
        "Weld Size (mm)": weld_size,
        "Weld Material Grade": material_grade,
        "Length of Weld (mm)": length_of_weld,
        "Strength of Connection (N)": connection_strength,
        "Plate 1 Yield Strength (N)": plate1_yield,
        "Plate 2 Yield Strength (N)": plate2_yield,
        "Efficiency of Connection": round(efficiency, 2)
    }
```

```python
# Example Input
P = 50  # kN
w = 200  # mm
t1 = 10  # mm
t2 = 8  # mm

# Run the design function
result = design_lap_joint(P, w, t1, t2)

# Print the results
for key, value in result.items():
    print(f"{key}: {value}")
```

2.6 Conclusion

This report outlines the design of a welded lap joint for two plates subjected to a tensile force, as per IS 800:2007. The design process includes material selection, strength calculations, and optimization to meet safety and efficiency standards. The designed joint will be evaluated based on various parameters to ensure it fulfills all design requirements.

2.7 References

1. IS 800:2007: General Construction in Steel – Code of Practice
2. IS 2062:2011: Steel for General Structural Purposes
3. *IS 800:2007 Code of Practice for General Construction in Steel – Code of Practice*
4. *IS 2062:2011 Steel for General Structural Purposes*

Chapter 3

Internship Task 1: OSI File Testing and Automation in Osdag

3.1 Internship Task 1: Problem Statement

The objective of Internship Task 1 is to develop a robust framework for validating `.osi` files used in Osdag, an open-source software for structural steel design. These JSON-based files contain parameters and configurations for structural steel connection modules, such as fin plate or base plate connections. The goal is to create an automated testing system that ensures valid `.osi` files pass silently while invalid files are flagged with appropriate error messages, enhancing the reliability of the Osdag software.

3.2 Internship Task 1: Tasks Done

To achieve the objectives of Task 1, you will implement a Python-based validation framework to automatically test `.osi` files. The following tasks will guide you through creating and organizing the necessary test files:

- **Develop Validation Logic**: Create a Python script (`osi_validate.py`) to validate `.osi` files. The script should check:
  - The presence of the mandatory `"module"` key in the JSON data.
  - The validity of the `"module"` value against a predefined list of supported modules (e.g., `finplate`, `baseplate`, `cleatangle`).

  - The type correctness of numeric fields, such as `beam_size`, ensuring they are integers or floats.

- **Write Test Cases**: Use the Pytest framework to create a test file (`test_osi_validate.py`) that tests the validation logic. The test cases should cover:
  - Valid `.osi` files that should pass all checks.
  - Invalid `.osi` files that should fail specific checks (e.g., missing `"module"` key or non-numeric `beam_size`).

- **Organize Test Files**: Structure the project directory to include separate folders for valid and invalid `.osi` files. This organization aids in managing test cases and ensures clarity during testing.

- **Automate Testing**: Create a batch script (`run_tests.bat`) for Windows to execute all test cases with a single command, streamlining the testing process.

3.2.1 Directory Structure

The following directory structure organizes the validation scripts and test files:

```
osi_testing
├─ osi_validate.py
├─ test_osi_validate.py
├─ valid_osi_files
│  ├─ Example_1.1.1.1.1.osi
│  └─ Example_1.1.1.1.2.osi
└─ invalid_osi_files
   ├─ Modified_Example_1.1.1.1.1.osi
   └─ Modified_Example_1.1.1.1.2.osi
```

3.2.2 Task Instructions: Writing Python Test Files

To write Python test files for validating `.osi` files, follow these steps:

1. **Set Up the Environment**:

- Ensure Python 3.8+ and Pytest are installed (`pip install pytest`).
- Create a project directory (`osi_testing`) with subfolders `valid_osi_files` and `invalid_osi_files`.
- Place sample `.osi` files in the respective folders. Valid files should conform to the expected JSON structure, while invalid files should include errors like missing keys or incorrect data types.

2. **Implement Validation Logic**:
   - Create `osi_validate.py` with functions to load and validate `.osi` files.
   - Use the `json` module to parse `.osi` files and implement checks for required keys, valid module names, and numeric fields.
3. **Write Pytest Test Cases**:
   - In `test_osi_validate.py`, write test functions using Pytest to verify each validation check.
   - Use `pytest.assert` to ensure the validation functions return expected results (e.g., `True` for valid files, `False` for invalid files).
   - Test both valid and invalid `.osi` files to cover all edge cases.
4. **Automate Test Execution**:
   - Create a batch script (`run_tests.bat`) with the command `pytest test_osi_validate.py --verbose` to run all tests.
   - Ensure the script is executable and points to the correct Python environment.

3.2.3 Example Validation Code

The following code demonstrates the validation logic for `.osi` files:

Listing 3.1: OSI Validation Logic

```
import json


def load_osi_file(file_path):
    """Load and parse a .osi file."""
```

```
    try:
        with open(file_path, 'r') as file:
            return json.load(file)
    except json.JSONDecodeError:
        raise ValueError(f"Invalid JSON format in {file_path}")

def validate_module_presence(data):
    """Check if the 'module' key exists in the data."""
    return "module" in data

def validate_known_module(data):
    """Check if the 'module' value is valid."""
    valid_modules = ["finplate", "baseplate", "cleatangle"]
    return data.get("module") in valid_modules

def validate_numeric_fields(data, field):
    """Check if a specified field is numeric."""
    return isinstance(data.get(field), (int, float))
```

3.2.4 Example Test Code

The following Pytest script tests the validation logic:

Listing 3.2: Pytest Test Cases

```
import pytest
from osi_validate import *

@pytest.fixture
def valid_osi_data():
    return load_osi_file("valid_osi_files/Example_1.1.1.1.1.osi")

@pytest.fixture
def invalid_osi_data():
    return load_osi_file("invalid_osi_files/Modified_Example_1
       .1.1.1.1.osi")
```

```
def test_module_key_present(valid_osi_data):
    assert validate_module_presence(valid_osi_data)

def test_valid_module_name(valid_osi_data):
    assert validate_known_module(valid_osi_data)

def test_beam_size_is_numeric(valid_osi_data):
    assert validate_numeric_fields(valid_osi_data, "beam_size")

def test_missing_module_key(invalid_osi_data):
    assert not validate_module_presence(invalid_osi_data)
```

Chapter 4

Internship Task 2: Validation Against Expected Parameters

4.1 Internship Task 2: Problem Statement

The goal of Internship Task 2 is to extend the validation framework to compare `.osi` file parameters (e.g., `beam_depth`, `column_size`) against expected values defined in an external reference spreadsheet. This ensures that `.osi` files adhere to pre-approved design standards, enhancing the accuracy of structural designs in Osdag.

4.2 Internship Task 2: Tasks Done

To validate `.osi` files against expected parameters, you will perform the following tasks:

- **Define Validation Functions**: Create functions in `osi_validate.py` to check if specific parameters match expected values from the reference spreadsheet.
- **Write Pytest Test Cases**: Develop test cases in `test_osi_validate.py` to verify each parameter against its expected value.
- **Map Expected Values**: Manually extract expected parameter values from the spreadsheet and hardcode them in the test cases for validation.

4.2.1 Task Instructions: Writing Python Test Files with Hardcoded Inputs

To write Python test files that validate `.osi` files against hardcoded expected inputs, follow these steps:

1. **Prepare the Environment**:
    - Ensure the project directory from Task 1 is set up, including `osi_validate.py` and `test_osi_validate.py`.
    - Obtain the reference spreadsheet containing expected parameter values for `.osi` files.
2. **Extract Expected Values**:
    - Review the spreadsheet to identify parameters (e.g., `beam_depth`, `column_size`) and their expected values.
    - Hardcode these values in the test file or a separate configuration file for reference.
3. **Extend Validation Logic**:
    - Add functions to `osi_validate.py` to compare `.osi` file parameters against hardcoded expected values.
    - Ensure the functions handle edge cases, such as missing parameters or type mismatches.
4. **Write Pytest Test Cases**:
    - In `test_osi_validate.py`, create test functions for each parameter, using `pytest.assert` to compare actual values from `.osi` files with expected values.
    - Use Pytest fixtures to load `.osi` files and reduce code duplication.

4.2.2 Example Validation Function

The following function validates the `beam_depth` parameter:

Listing 4.1: Checking Beam Depth

```
def check_beam_depth(data, expected_depth=300):
    """Check if beam_depth is an integer and meets the minimum
       value."""
    return isinstance(data.get("beam_depth"), int) and data["
       beam_depth"] >= expected_depth
```

4.2.3 Example Test Case

The following test case validates `beam_depth` against a hardcoded value:

Listing 4.2: Test for Beam Depth

```
import pytest
from osi_validate import *

@pytest.fixture
def osi_data():
    return load_osi_file("test_files/sample_design.osi")

def test_beam_depth(osi_data):
    assert check_beam_depth(osi_data, expected_depth=300)
```

Chapter 5

Internship Task 3: Integration with Conda Build Process

5.1 Internship Task 3: Problem Statement

The objective of Internship Task 3 is to integrate the OSI validation tests into the Conda package build process for Osdag. This ensures that all validation tests are executed automatically during packaging, improving code reliability and enabling compatibility with Continuous Integration/Continuous Deployment (CI/CD) pipelines.

5.2 Internship Task 3: Tasks Done

To integrate the validation tests with the Conda build process, you will perform the following tasks:

- **Set Up Version Control**:
  - Fork the Osdag repository from `https://github.com/osdag/osdag`.
  - Clone the forked repository to your local machine.
  - Create a new branch named `automation-testing` for your changes.
- **Develop Conda Recipe**: Create a Conda recipe with a `meta.yaml` file to define the package build process, including test execution.

- **Integrate Tests**: Configure the Conda recipe to run Pytest test cases during the build process, ensuring that build fails if any test case fails.
- **Commit and Push Changes**: Stage, commit, and push the testing scripts and Conda recipe to the `automation-testing` branch in your forked repository.

5.2.1 Task Instructions: Setting Up and Integrating with Conda

To integrate the OSI validation tests with the Conda build process, follow these detailed steps:

1. **Set Up the Repository**:
   - Fork the Osdag repository from `https://github.com/osdag/osdag` to your GitHub account.
   - Clone the forked repository to your local machine:

```
git clone https://github.com/<your-username>/osdag.git
cd osdag
```

   - Create a new branch for your changes:

```
git checkout -b feature/branch
```

2. **Create Conda Recipe**:
   - Create a directory named `osdag-conda-recipe` in your project.
   - Add a `meta.yaml` file to define the package metadata, build script, and test commands.
   - Ensure the recipe includes dependencies like `python`, `pytest`, and Osdag runtime requirements (e.g., `pyqt`, `numpy`).
3. **Integrate Test Execution**:
   - In the `meta.yaml` file, configure the `build: script:` section to install the package and run Pytest.
   - In the `test:` section, specify the test files and commands to execute Pytest.

- Ensure that test failures cause the Conda build to fail, enforcing quality control.

4. **Commit and Push Changes**:

   - Add the testing scripts (`osi_validate.py`, `test_osi_validate.py`) and Conda recipe files to the repository:

```
git add osi_validate.py test_osi_validate.py osdag-conda-
    recipe/
```

   - Commit the changes with a descriptive message:

```
git commit -m "Add OSI validation tests and Conda recipe
    for automated testing"
```

   - Push the `automation-testing` branch to your forked repository:

```
git push origin feature/branch
```

5. **Execute Conda Build**:

   - Navigate to the Conda recipe directory:

```
cd osdag-conda-recipe
```

   - Run the Conda build command:

```
conda build .
```

   - Verify that the build process executes the Pytest test cases and fails if any test does not pass.

5.2.2 Example Conda Recipe

The following `meta.yaml` file configures the Conda build process:

Listing 5.1: Conda Recipe for Testing

```
package:
  name: osdag
  version: "0.1"
```

```
source:
  path: ..

build:
  script:
    - pip install .
    - pytest --verbose --capture=no

requirements:
  build:
    - python >=3.8
    - pip
    - pytest
    - setuptools
    - setuptools_scm
  host:
    - python >=3.8
    - pip
    - pytest
    - setuptools
    - setuptools_scm
  run:
    - python >=3.8
    - pyqt
    - requests
    - numpy
    - pyyaml
    - pygithub
    - pytest

test:
  source_files:
    - recipe/tests/test_validator.py
```

```
    - recipe/tests/validator.py
  requires:
    - pytest
  commands:
    - pytest recipe/tests/


about:
  home: "https://osdag.fossee.in"
  summary: "Open Steel Design and Graphics (Osdag) - an open-
     source software for structural steel design developed by
     FOSSEE, IIT Bombay."
```

Chapter 6

Internship Task 4: Testing Tension Bolted Module with Mocks

6.1 Internship Task 4: Problem Statement

The goal of Internship Task 4 is to create a Python test suite to validate the `Tension_bolted` module in Osdag using the `unittest` framework and mocking techniques. Mocking is used to simulate the behavior of the `Tension_bolted` class, avoiding dependencies on its actual implementation, which may involve complex structural calculations or external resources.

6.1.1 Understanding Mocks

Mocks in Python, provided by the `unittest.mock` module, are fake implementations of objects or methods used in unit testing. They allow you to isolate the code under test by replacing dependencies (e.g., database calls, file operations, or complex computations) with controlled, predefined behaviors. In this task, you will mock the `Tension_bolted` class to return hardcoded results, enabling fast and reliable testing without executing the actual structural design logic.

- **Purpose of Mocks**: Mocks simulate the behavior of the `Tension_bolted` class, allowing you to test the logic that processes its outputs without running the actual computations.

- **How Mocks Are Used**: The `MagicMock` and `patch` utilities from `unittest.mock` are used to create a mock object for the `Tension_bolted` class and define its method outputs. For example, you can configure the `results_to_test` method to return a predefined dictionary of results.
- **Benefits**: Mocks make tests faster, independent of external systems, and easier to control by allowing you to specify expected outputs for specific inputs.

6.2 Internship Task 4: Tasks Done

To validate the `Tension_bolted` module, you will perform the following tasks:

- **Set Up the Test Environment**: Configure the test environment with necessary imports and a mock object for the `Tension_bolted` class.
- **Write Test Cases**: Create test cases using `unittest` to validate the outputs of the `Tension_bolted` module against expected values.
- **Use Mocks**: Mock the `Tension_bolted` class to return predefined results for various test cases, simulating different input scenarios.
- **Verify Outputs**: Assert that the mocked outputs match expected values for parameters like section designation, tension capacity, bolt grade, and number of bolts.

6.2.1 Task Instructions: Writing Tests with Mocks

To write a test suite for the `Tension_bolted` module using mocks, follow these steps:

1. **Set Up the Environment**:
    - Ensure Python 3.8+ and `unittest` are available (included in the standard library).
    - Create a test file named `test_tension_bolted_mock.py` in the `tests` directory of your Osdag repository.
    - Import necessary modules: `unittest`, `unittest.mock.MagicMock`, and `unittest.mock.pat`
2. **Configure the Mock**:

- Use `MagicMock` to create a mock object for the `Tension_bolted` class in the `setUp` method.
- Configure the mock to return predefined results for the `results_to_test` method and ensure `set_input_values` returns `None`.

3. **Write Test Cases**:
   - Define multiple test cases (e.g., `test_tension_bolted_test1`, `test_tension_bolted_test2`) to cover different input scenarios.
   - Use the `@patch` decorator to replace the real `Tension_bolted` class with the mock object during testing.
   - For each test case, define a `design_dict` with input parameters and a `mock_results` dictionary with expected outputs.
   - Assert that the mocked outputs match the expected values using `self.assertEqual` and `self.assertAlmostEqual` for floating-point comparisons.
4. **Run the Tests**:
   - Execute the test suite using:

```
python -m unittest test_tension_bolted_mock.py -v
```

   - Verify that all test cases pass, indicating that the mocked outputs align with the expected results.

6.2.2 Full Code

The following test suite validates the `Tension_bolted` module using mocks:

Listing 6.1: Test Suite for Tension Bolted Module with Mocks

```
import unittest
from unittest.mock import MagicMock, patch


# mock functions in python are fake implementations of real
    objects or methods.
#  t h e y re  used in unit testing when you  d o n t  want to depend
    on actual implementation
```

```
# (like database calls, file reads, or in this case, a heavy
    design computation class).
# the mock just pretends to do what the real method does and
    returns predefined values.
# this makes testing fast, isolated, and easy to control.

# here, we are mocking the Tension_bolted class (probably a heavy
     structural module from osdag),
# and we're faking its method outputs by giving hardcoded (
    predefined) results.

class TestTensionBolted(unittest.TestCase):
    def setUp(self):
        # create a fake object (mock) for the Tension_bolted
            class
        self.tension_bolted = MagicMock()

        # whenever results_to_test is called on our mock object,
            return an empty dictionary by default
        self.tension_bolted.results_to_test.return_value = {}

        # make sure set_input_values just silently returns
            nothing
        self.tension_bolted.set_input_values.return_value = None

    @patch('tension_bolted.Tension_bolted')  # this replaces the
       real class with a mock during the test
    def test_tension_bolted_test1(self, mock_tension_bolted):
        # test case 1: check if results match expected mock
            values

        # this is what we expect the real method to return (but
            we mock it)
        mock_results = {
```

```
        "KEY_DISP_DESIGNATION": "40 x 20 x 3",
        "KEY_DISP_TENSION_YIELDCAPACITY": "79.09",
        "KEY_OUT_DISP_BOLT_LINE": "2",
        "KEY_OUT_DISP_BOLTS_ONE_LINE": "2",
        "KEY_OUT_DISP_BOLT_CAPACITY": "17.71",
        "KEY_OUT_DISP_GRD_PROVIDED": "3.6",
        "KEY_OUT_DISP_D_PROVIDED": "20"
    }
    self.tension_bolted.results_to_test.return_value =
       mock_results

    # fake user input for the test
    design_dict = {
        "KEY_SECSIZE": "40 x 20 x 3",
        "KEY_GRD": "3.6",
        "KEY_D": "20",
        "KEY_MATERIAL": "E 250 (Fe 410 W)A",
        "KEY_SEC_PROFILE": "Angles",
        "KEY_LOCATION": "Long Leg",
        "KEY_TYP": "Bearing Bolt",
        "KEY_PLATETHK": "10",
        "KEY_AXIAL": "50",
    }

    # expected outputs based on the mock results
    expected_designation = "40 x 20 x 3"
    expected_tension_capacity_section = 79.09
    expected_tension_capacity_plate = 70.84
    expected_bolt_grade = 3.6
    expected_number_of_bolts = 4
    expected_bolt_diameter = 20

    # simulate setting the inputs and getting the result
    self.tension_bolted.set_input_values(design_dict)
```

```
        results = self.tension_bolted.results_to_test("temp_test1
            .txt")

        # assert that each value matches our expectations
        self.assertEqual(results["KEY_DISP_DESIGNATION"],
            expected_designation)
        self.assertAlmostEqual(float(results["
            KEY_DISP_TENSION_YIELDCAPACITY"]),
            expected_tension_capacity_section, places=2)

        # calculate bolt capacity manually and check if it
            matches expected plate capacity
        number_of_bolts = int(results["KEY_OUT_DISP_BOLT_LINE"])
            * int(results["KEY_OUT_DISP_BOLTS_ONE_LINE"])
        plate_capacity = float(results["
            KEY_OUT_DISP_BOLT_CAPACITY"]) * number_of_bolts
        self.assertAlmostEqual(plate_capacity,
            expected_tension_capacity_plate, places=2)
        self.assertEqual(float(results["KEY_OUT_DISP_GRD_PROVIDED
            "]), expected_bolt_grade)
        self.assertEqual(number_of_bolts,
            expected_number_of_bolts)
        self.assertEqual(int(results["KEY_OUT_DISP_D_PROVIDED"]),
             expected_bolt_diameter)

    @patch('tension_bolted.Tension_bolted')
    def test_tension_bolted_test2(self, mock_tension_bolted):
        # test case 2

        mock_results = {
            "KEY_DISP_DESIGNATION": "MC 100",
            "KEY_DISP_TENSION_YIELDCAPACITY": "104.82",
            "KEY_OUT_DISP_BOLT_LINE": "1",
            "KEY_OUT_DISP_BOLTS_ONE_LINE": "3",
```

```
            "KEY_OUT_DISP_BOLT_CAPACITY": "58.25",
            "KEY_OUT_DISP_GRD_PROVIDED": "12.9",
            "KEY_OUT_DISP_D_PROVIDED": "12"
        }
        self.tension_bolted.results_to_test.return_value =
            mock_results

        design_dict = {
            "KEY_SECSIZE": "MC 100",
            "KEY_GRD": "12.9",
            "KEY_D": "12",
            "KEY_MATERIAL": "E 250 (Fe 410 W)A",
            "KEY_SEC_PROFILE": "Channels",
            "KEY_LOCATION": "Web",
            "KEY_TYP": "Bearing Bolt",
            "KEY_PLATETHK": "10",
            "KEY_AXIAL": "50",
        }

        expected_designation = "MC 100"
        expected_tension_capacity_section = 104.82
        expected_tension_capacity_plate = 174.75
        expected_bolt_grade = 12.9
        expected_number_of_bolts = 3
        expected_bolt_diameter = 12

        self.tension_bolted.set_input_values(design_dict)
        results = self.tension_bolted.results_to_test("temp_test2
            .txt")

        self.assertEqual(results["KEY_DISP_DESIGNATION"],
            expected_designation)
        self.assertAlmostEqual(float(results["
            KEY_DISP_TENSION_YIELDCAPACITY"]),
```

```
            expected_tension_capacity_section, places=2)

        number_of_bolts = int(results["KEY_OUT_DISP_BOLT_LINE"])
            * int(results["KEY_OUT_DISP_BOLTS_ONE_LINE"])
        plate_capacity = float(results["
            KEY_OUT_DISP_BOLT_CAPACITY"]) * number_of_bolts
        self.assertAlmostEqual(plate_capacity,
            expected_tension_capacity_plate, places=2)
        self.assertEqual(float(results["KEY_OUT_DISP_GRD_PROVIDED
            "]), expected_bolt_grade)
        self.assertEqual(number_of_bolts,
            expected_number_of_bolts)
        self.assertEqual(int(results["KEY_OUT_DISP_D_PROVIDED"]),
             expected_bolt_diameter)

    @patch('tension_bolted.Tension_bolted')
    def test_tension_bolted_test3(self, mock_tension_bolted):
        # test case 3

        mock_results = {
            "KEY_DISP_DESIGNATION": "40 x 40 x 3",
            "KEY_DISP_TENSION_YIELDCAPACITY": "28.79",
            "KEY_OUT_DISP_BOLT_LINE": "2",
            "KEY_OUT_DISP_BOLTS_ONE_LINE": "2",
            "KEY_OUT_DISP_BOLT_CAPACITY": "12.0",
            "KEY_OUT_DISP_GRD_PROVIDED": "4.6",
            "KEY_OUT_DISP_D_PROVIDED": "10"
        }
        self.tension_bolted.results_to_test.return_value =
            mock_results

        design_dict = {
            "KEY_SECSIZE": "40 x 40 x 3",
            "KEY_GRD": "4.6",
```

```
            "KEY_D": "10",
            "KEY_MATERIAL": "E 250 (Fe 410 W)A",
            "KEY_SEC_PROFILE": "Angles",
            "KEY_LOCATION": "Long Leg",
            "KEY_TYP": "Bearing Bolt",
            "KEY_PLATETHK": "10",
            "KEY_AXIAL": "50",
        }

        expected_designation = "40 x 40 x 3"
        expected_tension_capacity_section = 28.79
        expected_tension_capacity_plate = 48.0
        expected_bolt_grade = 4.6
        expected_number_of_bolts = 4
        expected_bolt_diameter = 10

        self.tension_bolted.set_input_values(design_dict)
        results = self.tension_bolted.results_to_test("temp_test3
            .txt")

        self.assertEqual(results["KEY_DISP_DESIGNATION"],
            expected_designation)
        self.assertAlmostEqual(float(results["
            KEY_DISP_TENSION_YIELDCAPACITY"]),
            expected_tension_capacity_section, places=2)

        number_of_bolts = int(results["KEY_OUT_DISP_BOLT_LINE"])
            * int(results["KEY_OUT_DISP_BOLTS_ONE_LINE"])
        plate_capacity = float(results["
            KEY_OUT_DISP_BOLT_CAPACITY"]) * number_of_bolts
        self.assertAlmostEqual(plate_capacity,
            expected_tension_capacity_plate, places=2)
        self.assertEqual(float(results["KEY_OUT_DISP_GRD_PROVIDED
            "]), expected_bolt_grade)
```

```
        self.assertEqual(number_of_bolts,
            expected_number_of_bolts)
        self.assertEqual(int(results["KEY_OUT_DISP_D_PROVIDED"]),
            expected_bolt_diameter)

    @patch('tension_bolted.Tension_bolted')
    def test_tension_bolted_test4(self, mock_tension_bolted):
        # test case 4

        mock_results = {
            "KEY_DISP_DESIGNATION": "MC 175",
            "KEY_DISP_TENSION_YIELDCAPACITY": "363.86",
            "KEY_OUT_DISP_BOLT_LINE": "2",
            "KEY_OUT_DISP_BOLTS_ONE_LINE": "3",
            "KEY_OUT_DISP_BOLT_CAPACITY": "61.62",
            "KEY_OUT_DISP_GRD_PROVIDED": "5.6",
            "KEY_OUT_DISP_D_PROVIDED": "20"
        }
        self.tension_bolted.results_to_test.return_value =
            mock_results

        design_dict = {
            "KEY_SECSIZE": "MC 175",
            "KEY_GRD": "5.6",
            "KEY_D": "20",
            "KEY_MATERIAL": "E 250 (Fe 410 W)A",
            "KEY_SEC_PROFILE": "Channels",
            "KEY_LOCATION": "Web",
            "KEY_TYP": "Bearing Bolt",
            "KEY_PLATETHK": "10",
            "KEY_AXIAL": "50",
        }

        expected_designation = "MC 175"
```

```
        expected_tension_capacity_section = 363.86
        expected_tension_capacity_plate = 369.72
        expected_bolt_grade = 5.6
        expected_number_of_bolts = 6
        expected_bolt_diameter = 20

        self.tension_bolted.set_input_values(design_dict)
        results = self.tension_bolted.results_to_test("temp_test4
            .txt")

        self.assertEqual(results["KEY_DISP_DESIGNATION"],
            expected_designation)
        self.assertAlmostEqual(float(results["
            KEY_DISP_TENSION_YIELDCAPACITY"]),
            expected_tension_capacity_section, places=2)

        number_of_bolts = int(results["KEY_OUT_DISP_BOLT_LINE"])
            * int(results["KEY_OUT_DISP_BOLTS_ONE_LINE"])
        plate_capacity = float(results["
            KEY_OUT_DISP_BOLT_CAPACITY"]) * number_of_bolts
        self.assertAlmostEqual(plate_capacity,
            expected_tension_capacity_plate, places=2)
        self.assertEqual(float(results["KEY_OUT_DISP_GRD_PROVIDED
            "]), expected_bolt_grade)
        self.assertEqual(number_of_bolts,
            expected_number_of_bolts)
        self.assertEqual(int(results["KEY_OUT_DISP_D_PROVIDED"]),
             expected_bolt_diameter)

# this runs all tests when we execute the file
if __name__ == "__main__":
    unittest.main()
```

Chapter 7

Internship Task 5: Mimicking Fin Plate GUI Inputs

7.1 Internship Task 5: Problem Statement

The goal of Internship Task 5 is to create a Python test suite that mimics the Osdag GUI input process for the Fin Plate Connection module. The test suite should simulate the GUI's input validation logic, ensuring that input parameters are validated against database values and design preferences, and handle compatibility with legacy keys and material properties.

7.2 Internship Task 5: Tasks Done

To mimic the Fin Plate GUI inputs, you will perform the following tasks:

- **Set Up the Environment**: Configure the test environment with necessary imports, database connections, and paths to Osdag's SQLite database.
- **Implement Input Mimicry**: Create a function to simulate the GUI's input collection process, validating inputs against database values and handling custom material grades.
- **Write Test Cases**: Develop test cases to validate the input mimicry function with multiple `.osi` file scenarios.

- **Validate Outputs**: Ensure the generated design dictionary includes all required keys, validated values, and compatibility mappings.

7.2.1 Task Instructions: Mimicking Fin Plate GUI Inputs

To create a test suite that mimics the Fin Plate GUI inputs, follow these detailed steps:

1. **Set Up the Environment**:
    - Ensure Python 3.8+ and required Osdag dependencies (`sqlite3`, `pyqt`, `numpy`) are installed.
    - Verify that the Osdag SQLite database is accessible at `PATH_TO_DATABASE` (e.g., `path/to/osdag/ResourceFiles/Database/IntDesign.db`).
    - Create a test file named `test_fin_plate_mimicry.py` in the `tests` directory.
    - Import necessary modules: `sqlite3`, `FinPlateConnection`, `Common`, `MaterialValidator`, and predefined constants like `VALUES_CONN`, `VALUES_TYP`, `VALUES_GRD_CUSTOMIZED`, and `VALUES_PLATETHK_CUSTOMIZED`.
2. **Implement Input Mimicry Function**:
    - Create a function (`mimic_fin_plate_inputs`) that takes a test case dictionary and returns a validated design dictionary.
    - Connect to the Osdag database to fetch valid values for combobox fields (e.g., `KEY_SUPTNGSEC`, `KEY_D`).
    - Validate input fields (e.g., `KEY_SHEAR`, `KEY_GRD`) using appropriate validators (e.g., integer for `KEY_SHEAR`, float for `KEY_GRD`).
    - Handle custom material grades using `MaterialValidator` and compute `fu` and `fy` based on thickness.
    - Map legacy compatibility keys to ensure backward compatibility with older Osdag versions.
3. **Write Test Cases**:
    - Define multiple test cases in `test_mimic_fin_plate_inputs`, each representing a different `.osi` file scenario.

- For each test case, call `mimic_fin_plate_inputs` and verify that the output design dictionary is correctly populated.
- Use exception handling to catch and report validation errors.

4. **Run the Tests**:
   - Execute the test suite using:

```
python test_fin_plate_mimicry.py
```

   - Verify that all test cases pass, indicating successful validation of the input mimicry process.

7.2.2 Full Code

The following test suite mimics the Fin Plate GUI inputs:

Listing 7.1: Test Suite for Mimicking Fin Plate GUI Inputs

```
import sqlite3
from design_type.connection.fin_plate_connection import
    FinPlateConnection
from utils.common.Common import connectdb, connectdb1,
    MaterialValidator, VALUES_CONN, VALUES_TYP,
    VALUES_GRD_CUSTOMIZED, VALUES_PLATETHK_CUSTOMIZED
from utils.common.component import PATH_TO_DATABASE


# function to get ultimate (fu) and yield (fy) strengths for a
    material grade
def get_fy_fu(material_grade, thickness=None):
    """return fu and fy for a material grade, considering
        thickness for plates."""
    # handle standard material grade e 250 (fe 410 w)a
    if material_grade == "E 250 (Fe 410 W)A":
        # if no thickness provided, return default fu, fy
        if thickness is None:
            return 410, 250
        else:
```

```
            # convert thickness to float for comparison
            thickness = float(thickness)
            # return fu, fy based on thickness ranges
            if thickness <= 20:
                return 410, 250, 250, 250
            elif thickness <= 40:
                return 410, 250, 240, 240
            else:
                return 410, 250, 240, 230
    # handle custom material grades starting with "cus_"
    elif material_grade.startswith("Cus_"):
        validator = MaterialValidator(material_grade)
        # check if custom material is valid
        if validator.is_valid_custom():
            parts = material_grade.split('_')
            # extract fu and fy from custom grade name
            fu, fy = float(parts[-1]), float(parts[-2])
            # return fu, fy based on thickness
            if thickness is None:
                return fu, fy
            else:
                return fu, fy, fy, fy
    # default fu, fy if material grade is unknown
    return 410, 250

# function to mimic gui input for fin plate connection
def mimic_fin_plate_inputs(test_case_data):
    """mimic gui input collection for fin plate connection,
       creating a design_dictionary."""
    # copy input data to avoid modifying original
    design_dictionary = test_case_data.copy()

    # connect to database to fetch valid values
    conn = sqlite3.connect(PATH_TO_DATABASE)
```

```
    cursor = conn.cursor()

    # define input fields for fin plate connection
    input_fields = [
        {"key": "KEY_MODULE", "type": "TYPE_MODULE", "value": "
           Fin Plate Connection"},
        {"key": "KEY_CONN", "type": "TYPE_COMBOBOX", "values":
           VALUES_CONN},
        # fetch beams or columns based on connection type
        {"key": "KEY_SUPTNGSEC", "type": "TYPE_COMBOBOX",
         "values": connectdb("Beams") if test_case_data.get("
            KEY_CONN") == "Beam-Beam" else connectdb("Columns")},
        {"key": "KEY_SUPTNGSEC_MATERIAL", "type": "TYPE_COMBOBOX"
           , "values": connectdb("Material")},
        {"key": "KEY_SUPTDSEC", "type": "TYPE_COMBOBOX", "values"
           : connectdb("Beams")},
        {"key": "KEY_SUPTDSEC_MATERIAL", "type": "TYPE_COMBOBOX",
            "values": connectdb("Material")},
        {"key": "KEY_SHEAR", "type": "TYPE_TEXTBOX", "validator":
            "Int Validator"},
        {"key": "KEY_AXIAL", "type": "TYPE_TEXTBOX", "validator":
            "Int Validator"},
        {"key": "KEY_D", "type": "TYPE_COMBOBOX_CUSTOMIZED", "
           values": connectdb1()},
        {"key": "KEY_TYP", "type": "TYPE_COMBOBOX", "values":
           VALUES_TYP},
        {"key": "KEY_GRD", "type": "TYPE_COMBOBOX_CUSTOMIZED", "
           values": VALUES_GRD_CUSTOMIZED},
        {"key": "KEY_PLATETHK", "type": "TYPE_COMBOBOX_CUSTOMIZED
           ", "values": VALUES_PLATETHK_CUSTOMIZED},
        {"key": "KEY_CONNECTOR_MATERIAL", "type": "TYPE_COMBOBOX"
           , "values": connectdb("Material")},
    ]
```

```
    # define design preference fields
    design_pref_fields = [
        {"key": "KEY_DP_BOLT_TYPE", "type": "TYPE_COMBOBOX", "
           values": ["Pretensioned", "Non pre-tensioned"]},
        {"key": "KEY_DP_BOLT_HOLE_TYPE", "type": "TYPE_COMBOBOX",
            "values": ["Standard", "Over-sized"]},
        {"key": "KEY_DP_BOLT_SLIP_FACTOR", "type": "TYPE_TEXTBOX"
           , "value": "0.3"},
        {"key": "KEY_DP_WELD_FAB", "type": "TYPE_COMBOBOX", "
           values": ["Shop Weld", "Field Weld"]},
        {"key": "KEY_DP_WELD_MATERIAL_G_O", "type": "TYPE_TEXTBOX
           ", "value": "410"},
        {"key": "KEY_DP_DETAILING_EDGE_TYPE", "type": "
           TYPE_COMBOBOX",
         "values": ["Sheared or hand flame cut", "Rolled, machine
            -flame cut, sawn and planed"]},
        {"key": "KEY_DP_DETAILING_GAP", "type": "TYPE_TEXTBOX", "
           value": "10"},
        {"key": "KEY_DP_DETAILING_CORROSIVE_INFLUENCES", "type":
           "TYPE_COMBOBOX", "values": ["No", "Yes"]},
        {"key": "KEY_DP_DESIGN_METHOD", "type": "TYPE_COMBOBOX",
           "values": ["Limit State Design"]},
    ]

    # validate each input field
    for field in input_fields:
        key = field["key"]
        input_type = field["type"]
        valid_values = field.get("values", [])

        value = design_dictionary.get(key)

        # check combobox values are valid
        if input_type == "TYPE_COMBOBOX":
```

```
            if value not in valid_values and value not in ["",
                None]:
                raise ValueError(f"Invalid value '{value}' for {
                    key}. Valid: {valid_values}")
        # check customized combobox values
        elif input_type == "TYPE_COMBOBOX_CUSTOMIZED":
            if isinstance(valid_values, list) and valid_values
                and isinstance(valid_values[0], list):
                valid_values = valid_values[0]
            if value not in valid_values and value not in ["",
                None]:
                raise ValueError(f"Invalid value '{value}' for {
                    key}. Valid: {valid_values}")
        # validate textbox inputs as integers
        elif input_type == "TYPE_TEXTBOX" and field.get("
            validator") == "Int Validator":
            try:
                if value not in ["", None]:
                    int_value = int(value)
                    if int_value <= 0:
                        print(f"Warning: {key} must be positive.
                            Using default: 1")
                        design_dictionary[key] = "1"
                else:
                    design_dictionary[key] = ""
            except ValueError:
                print(f"Error: Invalid {key}: '{value}'. Using
                    default: 1")
                design_dictionary[key] = "1"

        # validate custom material grades
        if key.endswith("_MATERIAL") and value and value.
            startswith("Cus_"):
            validator = MaterialValidator(value)
```

```
            if not validator.is_valid_custom():
                print(f"Warning: Invalid custom material '{value
                    }' for {key}. Using default: E 250 (Fe 410 W)A
                    ")
                design_dictionary[key] = "E 250 (Fe 410 W)A"

        # convert specific keys to float or string
        if key == "KEY_GRD":
            design_dictionary[key] = float(value) if value else
                8.8
        elif key == "KEY_D":
            design_dictionary[key] = float(value) if value else
                20.0
        else:
            design_dictionary[key] = str(value) if value is not
                None else ""

    # validate design preference fields
    for field in design_pref_fields:
        key = field["key"]
        input_type = field["type"]
        valid_values = field.get("values", [])
        default_value = field.get("value", valid_values[0] if
            valid_values else "")

        value = design_dictionary.get(key, default_value)

        # check design preference combobox values
        if input_type == "TYPE_COMBOBOX":
            if value not in valid_values:
                print(f"Warning: Invalid value '{value}' for {key
                    }. Using default: {default_value}")
                design_dictionary[key] = default_value
        # validate textbox inputs as floats
```

```
        elif input_type == "TYPE_TEXTBOX":
            try:
                float(value)
            except ValueError:
                print(f"Warning: Invalid value '{value}' for {key
                    }. Using default: {default_value}")
                design_dictionary[key] = default_value

    # add compatibility keys for legacy support
    compatibility_keys = {
        "Connectivity *": "KEY_CONN",
        "Member.Supporting_Section.Designation" : "KEY_SUPTNGSEC"
            ,
        "Member.Supported_Section.Designation": "KEY_SUPTDSEC",
        "Member.Supporting_Section.Material": "
            KEY_SUPTNGSEC_MATERIAL",
        "Member.Supported_Section.Material": "
            KEY_SUPTDSEC_MATERIAL",
        "Load.Shear": "KEY_SHEAR",
        "Load.Axial": "KEY_AXIAL",
        "Bolt.Diameter": "KEY_D",
        "Bolt.Grade": "KEY_GRD",
        "Bolt.Type": "KEY_TYP",
        "Bolt.Bolt_Hole_Type": "KEY_DP_BOLT_HOLE_TYPE",
        "Bolt.TensionType": "KEY_DP_BOLT_TYPE",
        "Detailing.Bolt_Slip_Factor": "KEY_DP_BOLT_SLIP_FACTOR",
        "Detailing.Edge_type": "KEY_DP_DETAILING_EDGE_TYPE",
        "Detailing.Corrosive_Influences": "
            KEY_DP_DETAILING_CORROSIVE_INFLUENCES",
        "Detailing.Gap": "KEY_DP_DETAILING_GAP",
        "Design.Method": "KEY_DP_DESIGN_METHOD",
        "Material": "KEY_CONNECTOR_MATERIAL",
        "Module": "KEY_MODULE",
        "Plate.Thickness": "KEY_PLATETHK",
```

```
        "Plate.Material_Grade": "KEY_CONNECTOR_MATERIAL",
        "Connector.Material": "KEY_CONNECTOR_MATERIAL",
        "Shear_Force": "KEY_SHEAR",
        "Axial_Force": "KEY_AXIAL",
        "Weld.Material": "KEY_DP_WELD_MATERIAL_G_O",
        "Weld.Material_Grade_OverWrite": "
            KEY_DP_WELD_MATERIAL_G_O",
        "Weld.Fab": "KEY_DP_WELD_FAB",
        "Weld.Fabrication": "KEY_DP_WELD_FAB",
        "Weld.Fu": "KEY_DP_WELD_MATERIAL_G_O",
    }

    # map compatibility keys to design dictionary
    for dest_key, src_key in compatibility_keys.items():
        design_dictionary[dest_key] = design_dictionary.get(
            src_key, "")

    # calculate material properties for supporting, supported,
       and connector
    material_keys = [
        ("KEY_SUPTNGSEC_FU", "KEY_SUPTNGSEC_FY", "
            KEY_SUPTNGSEC_MATERIAL", None),
        ("KEY_SUPTDSEC_FU", "KEY_SUPTDSEC_FY", "
            KEY_SUPTDSEC_MATERIAL", None),
        ("KEY_CONNECTOR_FU", "KEY_CONNECTOR_FY_20", "
            KEY_CONNECTOR_MATERIAL", design_dictionary.get('
            KEY_PLATETHK', 10)),
    ]

    # assign fu and fy based on material and thickness
    for fu_key, fy_key, mat_key, thickness in material_keys:
        material = design_dictionary.get(mat_key, "E 250 (Fe 410
            W)A")
        if thickness is None:
```

```
            fu, fy = get_fy_fu(material)
            design_dictionary[fu_key] = str(fu)
            design_dictionary[fy_key] = str(fy)
        else:
            fu, fy_20, fy_20_40, fy_40 = get_fy_fu(material,
                float(thickness))
            design_dictionary[fu_key] = str(fu)
            design_dictionary[fy_key] = str(fy_20)
            design_dictionary['KEY_CONNECTOR_FY_20_40'] = str(
                fy_20_40)
            design_dictionary['KEY_CONNECTOR_FY_40'] = str(fy_40)

    # close database connection
    conn.close()
    return design_dictionary

# function to test fin plate input mimicry with four test cases
def test_mimic_fin_plate_inputs():
    """test the mimicry function with four osi file test cases.
       """
    # define four test cases based on osi files
    test_cases = [
        # finplatetest1.osi
        {
            "KEY_MODULE": "Fin Plate Connection",
            "KEY_CONN": "Column Flange-Beam Web",
            "KEY_SUPTNGSEC": "HB 300",
            "KEY_SUPTNGSEC_MATERIAL": "E 250 (Fe 410 W)A",
            "KEY_SUPTDSEC": "MB 200",
            "KEY_SUPTDSEC_MATERIAL": "E 250 (Fe 410 W)A",
            "KEY_SHEAR": "5",
            "KEY_AXIAL": "",
            "KEY_D": "20",
            "KEY_GRD": "10.9",
```

```
        "KEY_TYP": "Bearing Bolt",
        "KEY_PLATETHK": "10",
        "KEY_CONNECTOR_MATERIAL": "E 250 (Fe 410 W)A",
        "KEY_DP_BOLT_TYPE": "Pretensioned",
        "KEY_DP_BOLT_HOLE_TYPE": "Standard",
        "KEY_DP_BOLT_SLIP_FACTOR": "0.3",
        "KEY_DP_WELD_FAB": "Shop Weld",
        "KEY_DP_WELD_MATERIAL_G_O": "410",
        "KEY_DP_DETAILING_EDGE_TYPE": "Sheared or hand flame
           cut",
        "KEY_DP_DETAILING_GAP": "10",
        "KEY_DP_DETAILING_CORROSIVE_INFLUENCES": "No",
        "KEY_DP_DESIGN_METHOD": "Limit State Design"
    },
    # finplatetest2.osi
    {
        "KEY_MODULE": "Fin Plate Connection",
        "KEY_CONN": "Column Web-Beam Web",
        "KEY_SUPTNGSEC": "HB 200",
        "KEY_SUPTNGSEC_MATERIAL": "E 250 (Fe 410 W)A",
        "KEY_SUPTDSEC": "MB 200",
        "KEY_SUPTDSEC_MATERIAL": "E 250 (Fe 410 W)A",
        "KEY_SHEAR": "5",
        "KEY_AXIAL": "",
        "KEY_D": "20",
        "KEY_GRD": "8.8",
        "KEY_TYP": "Friction Grip Bolt",
        "KEY_PLATETHK": "16",
        "KEY_CONNECTOR_MATERIAL": "E 250 (Fe 410 W)A",
        "KEY_DP_BOLT_TYPE": "Pretensioned",
        "KEY_DP_BOLT_HOLE_TYPE": "Standard",
        "KEY_DP_BOLT_SLIP_FACTOR": "0.3",
        "KEY_DP_WELD_FAB": "Shop Weld",
        "KEY_DP_WELD_MATERIAL_G_O": "410",
```

```
        "KEY_DP_DETAILING_EDGE_TYPE": "Sheared or hand flame
           cut",
        "KEY_DP_DETAILING_GAP": "10",
        "KEY_DP_DETAILING_CORROSIVE_INFLUENCES": "No",
        "KEY_DP_DESIGN_METHOD": "Limit State Design"
    },
    # finplatetest3.osi
    {
        "KEY_MODULE": "Fin Plate Connection",
        "KEY_CONN": "Beam-Beam",
        "KEY_SUPTNGSEC": "MB 200",
        "KEY_SUPTNGSEC_MATERIAL": "E 250 (Fe 410 W)A",
        "KEY_SUPTDSEC": "MB 300",
        "KEY_SUPTDSEC_MATERIAL": "E 250 (Fe 410 W)A",
        "KEY_SHEAR": "50",
        "KEY_AXIAL": "8",
        "KEY_D": "20",
        "KEY_GRD": "8.8",
        "KEY_TYP": "Bearing Bolt",
        "KEY_PLATETHK": "12",
        "KEY_CONNECTOR_MATERIAL": "E 250 (Fe 410 W)A",
        "KEY_DP_BOLT_TYPE": "Pretensioned",
        "KEY_DP_BOLT_HOLE_TYPE": "Standard",
        "KEY_DP_BOLT_SLIP_FACTOR": "0.3",
        "KEY_DP_WELD_FAB": "Shop Weld",
        "KEY_DP_WELD_MATERIAL_G_O": "410",
        "KEY_DP_DETAILING_EDGE_TYPE": "Sheared or hand flame
           cut",
        "KEY_DP_DETAILING_GAP": "10",
        "KEY_DP_DETAILING_CORROSIVE_INFLUENCES": "No",
        "KEY_DP_DESIGN_METHOD": "Limit State Design"
    },
    # finplatetest4.osi
    {
```

```
            "KEY_MODULE": "Fin Plate Connection",
            "KEY_CONN": "Beam-Beam",
            "KEY_SUPTNGSEC": "MB 300",
            "KEY_SUPTNGSEC_MATERIAL": "E 250 (Fe 410 W)A",
            "KEY_SUPTDSEC": "MB 200",
            "KEY_SUPTDSEC_MATERIAL": "E 250 (Fe 410 W)A",
            "KEY_SHEAR": "50",
            "KEY_AXIAL": "8",
            "KEY_D": "20",
            "KEY_GRD": "8.8",
            "KEY_TYP": "Bearing Bolt",
            "KEY_PLATETHK": "20",
            "KEY_CONNECTOR_MATERIAL": "E 250 (Fe 410 W)A",
            "KEY_DP_BOLT_TYPE": "Non pre-tensioned",
            "KEY_DP_BOLT_HOLE_TYPE": "Over-sized",
            "KEY_DP_BOLT_SLIP_FACTOR": "0.3",
            "KEY_DP_WELD_FAB": "Field Weld",
            "KEY_DP_WELD_MATERIAL_G_O": "410",
            "KEY_DP_DETAILING_EDGE_TYPE": "Rolled, machine-flame
               cut, sawn and planed",
            "KEY_DP_DETAILING_GAP": "10",
            "KEY_DP_DETAILING_CORROSIVE_INFLUENCES": "No",
            "KEY_DP_DESIGN_METHOD": "Limit State Design"
        }
    ]

    # run each test case and validate
    for i, test_case in enumerate(test_cases, 1):
        print(f"\nRunning Test Case {i} (FinPlateTest{i}.osi)")
        try:
            design_dict = mimic_fin_plate_inputs(test_case)
            print("Design Dictionary:")
            for key, value in sorted(design_dict.items()):
                print(f"  {key}: {value}")
```

```
            # create fin plate connection object and set inputs
            fin_plate = FinPlateConnection()
            fin_plate.set_input_values(design_dict)
            print(f"Test Case {i}: Validation Successful")
        except Exception as e:
            print(f"Test Case {i}: Validation Failed - {str(e)}")
            raise

# run tests if script is executed directly
if __name__ == "__main__":
    test_mimic_fin_plate_inputs()
```

7.3 Documentation and Contribution

7.3.1 GitHub Repository Links

- Testing Repository: `https://github.com/lakshanashreee/Osdag/tree/testing-framework-p`
- Conda Recipe Repository: `https://github.com/lakshanashreee/osdag-conda-recipe/tree/feature-branch`

7.3.2 Summary

- Developed a validation framework for `.osi` files, ensuring structural and semantic correctness.
- Extended the framework to validate parameters against hardcoded expected values from a reference spreadsheet.
- Created test suites for the `Tension_bolted` module using mocks to simulate behavior.
- Implemented a test suite to mimic Fin Plate GUI inputs, validating against database values and handling legacy compatibility.

Chapter 8

Appendix

8.1 Work Reports

Internship Work Report			
Name:	Lakshana Shree S		
Team:	Osdag		
Internship:	Semester Long Internship 2025		
Date	**Day**	**Task**	**Hours Worked**
13-02-2025	Thursday	Installation of Osdag [Not completed].	6
14-02-2025	Friday	Installation of Osdag [Completed] and did initial testing.	5
15-02-2025	Saturday	Explored the code files [Task-0]	4
16-02-2025	Sunday	Worked on the report [Task-0]	5
17-02-2025	Monday	Attended meeting and then explored the code files again	4
18-02-2025	Tuesday	Learning how osdag works	5
19-02-2025	Wednesday	GitHub Discipline Meet and revised all the content taught	4
20-02-2025	Thursday	Study Break	0
21-02-2025	Friday	CLI & Unit Testing meeting & started working on the task	4
22-02-2025	Saturday	Study Break	0
23-02-2025	Sunday	Downloaded all the osi files from sample design	4
24-02-2025	Monday	Imported and tested the osi files on osdag gui	6
25-02-2025	Tuesday	Created modified OSI files with intentional errors.	4
26-02-2025	Wednesday	Wrote a batch script to automate to test the osi files.	5
27-02-2025	Thursday	Did the initial testing and execution.	4
28-02-2025	Friday	Developed a Python script to detect and specify error.	5
01--03-2025	Saturday	Ran multiple OSI files through the validation process.	6
02--03-2025	Sunday	Setting (overview) of the task.	4
03--03-2025	Monday	Debugged and improved error detection for accuracy.	4
04--03-2025	Tuesday	Finalized the process, ensured all scripts function correctly.	5
05--03-2025	Wednesday	Worked further on the task and had a meeting to review the ta	4.5
06--03-2024	Thursday	Presented my task and worked on the correction.	5
07--03-2025	Friday	Started working on the modified task.	5
08--03-2025	Saturday	Finalized my modified code.	4
09--03-2025	Sunday	Automated the testing process.	4
10–03-2025	Monday	Worked on the accuracy.	4

11--03-2025	Tuesday	Finalized my modified task and pushed the code to my git rep	5
12--03-2025	Wednesday	Had a meeting to present my modified task & worked on the c	4
13--03-2025	Thursday	Study Break	0
14--03-2025	Friday	Study Break	0
15--03-2025	Saturday	Study Break	0
16--03-2025	Sunday	Study Break	0
17--03-2025	Monday	Study Break	0
18--03-2025	Tuesday	Attended the meeting and started working on the insights.	6
19--03-2025	Wednesday	Explored about writing test in python.	4
20--03-2025	Thursday	Testing the test files written in python.	4
21–03-2025	Friday	Installed FreeCAD explored their interfaces.	5
22–03-2025	Saturday	Explored about FreeCAD.	5
23–03-2025	Sunday	Learning how to write test in Free CAD.	4
24–03-2025	Monday	Reviewed FreeCAD's Testing Framework to understand Free	5
25-03-2025	Tuesday	Meeting with mentors and working on the corrections	5
26-03-2025	Wednesday	Studied pytest detailly	4
27-03-2025	Thursday	Learned how to write test files using pytest	6
28-03-2025	Friday	Wrote test files using pytest	6
29-03-2025	Saturday	Created a repo in GITHUB and arranged all the files neatly	4
30-03-2025	Sunday	Was waiting for the OSI files to proceed further	0
31-03-2025	Monday	Was waiting for the OSI files to proceed further	0
01--04-2025	Tuesday	Was waiting for the OSI files to proceed further	0
02--04-2025	Wednesday	Had a meeting and Got the OSI files and the excel sheet valu	7
03--04-2025	Thursday	Read through the files (related to pytest) my mentor sent	6
04--04-2025	Friday	Started working on the newly assigned task. (to write tests for	7
05--04-2025	Saturday	Forked the Osdag repo and cloned it. Then, created a new br	6
06--04-2025	Sunday	Worked on writing the tests for those OSI files.	7
07--04-2025	Monday	Study break	0
08--04-2025	Tuesday	Study break	0
09--04-2025	Wednesday	Study break	0
10--04-2025	Thursday	Study break	0

11--04-2025	Friday	Study break	0
12--04-2025	Saturday	Study break	0
13--04-2025	Sunday	Study break	0
14--04-2025	Monday	Study break	0
15--04-2025	Tuesday	Meeting with mentors and started working on the corrections	4
16-04-2025	Wednesday	Started modifying my tests using pytest with proper assert sta	5
17-04-2025	Thursday	Started exploring my new task (runs during pip and conda ins	4
18-04-2025	Friday	Created a setup.py file to try running my tests	7
19-04-2025	Saturday	Worked on automating tests during conda build	7
20-04-2025	Sunday	Studied about toml formatted files	5
21-04-2025	Monday	Study break	0
22-04-2025	Tuesday	Study break	0
23-04-2025	Wednesday	Study break	0
24-04-2025	Thursday	Study break	0
25-04-2025	Friday	setup.py file to define a custom install command that automat	6
26-04-2025	Saturday	Had a meeting and got a new task (testing during conda build	4
27-04-2025	Sunday	Then deleted setup.py file and explored about pyproject.toml	7
28-04-2025	Monday	Had a meet and explored how to run the tests when making tl	7
29-04-2025	Tuesday	Forked the Osdag-Conda-Recipe and cloned it and created a	8
30-04-2025	Wednesday	Study break	0
1-5-2025	Thursday	Study break	0
02-05-2025	Friday	Study break	0
03-05-2025	Saturday	Study break	0
04-05-2025	Sunday	Study break	0
05-05-2025	Monday	Study break	0
06-05-2025	Tuesday	Study break	0
07-05-2025	Wednesday	Study break	0
08-05-2025	Thursday	Study break	0
09-05-2025	Friday	Study break	0
10-05-2025	Saturday	Study break	0
11-05-2025	Sunday	Study break	0

12-05-2025	Monday	Study break	0
13-05-2025	Tuesday	Worked on the automated tests while conda build	9
14-05-2025	Wednesday	Had a 1 hour meet with the installer team and relfected on the	6
15-05-2025	Thursday	Started working on the documentation: testing modules	5
16-05-2025	Friday	Had a meet with the unit testing team and worked on the insig	4
17-05-2025	Saturday	Worked on the documentation: testing modules	5
18-05-2025	Sunday	Finished working on the document: testing modules	5
19-05-2025	Monday	Finished all the minor pending changes: change branch name	6
20-05-2025	Tuesday	Worked on the requested changes of my mentor in document	5
21-05-2025	Wednesday	Had a meet with the new interns and worked on the insights.	6
22-05-2025	Thursday	Worked on rearranging my git repo for better usage for the ne	4
23-05-2025	Friday	Finished working on the requested changes in the document:	6
24-05-2025	Saturday	Worked on the conda build task: trying resolving the prolonge	5
25-05-2025	Sunday	Worked on the conda build task: trying resolving the prolonge	4
26-05-2025	Monday	Finished the conda build task.	5
27-05-2025	Tuesday	Had a meet with the unit testing team and got a new task, to v	5
28-05-2025	Wednesday	Started working on the new task that is to create a unit test to	7
29-05-2025	Thursday	Working on the new task: unit test	6
30-05-2025	Friday	Working on the new task: unit test	5
31-05-2025	Saturday	Working on the new task: unit test	7
01-06-2025	Sunday	Working on the new task: unit test	4
02-06-2025	Monday	Had a meeting and a review of my work: started worked on th	6
03-06-2025	Tuesday	Working on the modifed task: unit test	5
04-06-2025	Wednesday	Had a meeting and a review of my work: started worked on th	6
05-06-2025	Thursday	Working on the modifed task: unit test	5
06-06-2025	Friday	Had a meeting with Aum, tried troubleshooting the errors i fac	6
07-06-2025	Saturday	Worked on the tradition approach: fin plate unit testing	5
08-06-2025	Sunday	Worked on the tradition approach: fin plate unit testing	6
09-06-2025	Monday	Got a lots of errors while working on the traditional approach	10
10-06-2025	Tuesday	Had a meeting with Aum, tried troubleshooting the errors i fac	4
11-06-2025	Wednesday	Worked on the tradition approach: fin plate unit testing	6

12-06-2025	Thursday	Had a meeting with Aum and was assigned to work on a new	6
13-06-2025	Friday	Worked on the mimicry approach from scratch	10
14-06-2025	Saturday	2 osi files were validated successfully but two failed, we had a	4
15-06-2025	Sunday	Tried to make the remaining 2 osi files validate too	5
16-06-2025	Monday	Resolved some database path issues to make the remaining	6
17-06-2025	Tuesday	Tried to make the remaining 2 osi files validate too	6
18-06-2025	Wednesday	Tried to debug the errors I got	4
19-06-2025	Thursday	Finally, debugged that 4 of my osi files validate successfully	4
20-06-2025	Friday	Had a meet with my mentor explaning my current progress	4
21-06-2025	Saturday	Worked on mimicing the calculations part	6
22-06-2025	Sunday	Worked on mimicing the calculations part	4
23-06-2025	Monday	Worked on the report and the calculations part	7
24-06-2025	Tuesday	Worked on the report	4
25-06-2025	Wednesday	Had a meet with my mentor for discussing my current git repo	5
26-06-2025	Thursday	Worked on the report	5

Chapter 9

Conclusions

9.1 Tasks Accomplished

The internship focused on enhancing the reliability and automation of Osdag's OSI file validation process. The following tasks were successfully completed:

- **Task 1: OSI File Validation Framework**: Developed a Python-based framework (`osi_validate.py`) to validate `.osi` files, checking for the presence of required keys, valid module names, and correct numeric field types. Created Pytest test cases (`test_osi_validate.py`) to verify valid and invalid files, organized test files into `valid_osi_files` and `invalid_osi_files` folders, and implemented a batch script (`run_tests.bat`) for one-click test execution.

- **Task 2: Parameter Validation Against Expected Values**: Extended the validation framework to compare `.osi` file parameters (e.g., `beam_depth`, `column_size`) against hardcoded values from a reference spreadsheet. Implemented validation functions and corresponding Pytest test cases to ensure compliance with design standards.

- **Task 3: Conda Build Integration**: Integrated the OSI validation tests into the Conda package build process by creating a `meta.yaml` file. Configured the recipe to run Pytest during the build, ensuring test failures halt the process. Forked the Osdag repository, created an `automation-testing` branch, and pushed changes to the forked repository.

- **Task 4: Testing Tension Bolted Module with Mocks**: Developed a `unittest`-based test suite (`test_tension_bolted_mock.py`) to validate the `Tension_bolted` module using mocks. Simulated the module's behavior with predefined outputs to test parameters like section designation, tension capacity, and bolt properties.

- **Task 5: Mimicking Fin Plate GUI Inputs**: Created a test suite (`test_fin_plate_mimicry.py`) to mimic the Osdag GUI input process for the Fin Plate Connection module. Validated inputs against database values, handled custom material grades, and ensured compatibility with legacy keys.

9.2 Skills Developed

The internship provided opportunities to develop a range of technical and professional skills, including:

- **Technical Skills**:
  - **Python Programming**: Gained proficiency in writing modular, reusable Python code for validation and testing, using libraries like `json`, `pytest`, `unittest`, and `unittest.mock`.
  - **Testing Frameworks**: Mastered Pytest and `unittest` for creating automated test suites, including the use of fixtures and mocks to isolate dependencies.
  - **Conda Packaging**: Learned to create and configure Conda recipes (`meta.yaml`) for package building and test integration, enhancing CI/CD compatibility.
  - **Database Interaction**: Developed skills in querying SQLite databases to validate input parameters and retrieve design data.
  - **Version Control**: Improved expertise in Git and GitHub workflows, including forking, branching, committing, and pushing changes to a repository.
- **Professional Skills**:
  - **Problem-Solving**: Tackled complex validation challenges, designing robust solutions to ensure software reliability.

- **Documentation**: Enhanced technical writing skills by creating clear, detailed documentation for code, test cases, and processes.
- **Time Management**: Managed multiple tasks within deadlines, prioritizing deliverables to meet project goals.
- **Attention to Detail**: Ensured precision in validation logic and test case design to catch errors and maintain high-quality standards.

Bibliography

[1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.

[2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.

[3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.