FOSSEE Summer Fellowship Report

On

Unit Testing – Testing Framework for Osdag

Submitted by

Jerin Jiss George

4th Year B.Tech Student, School of Computing Science Engineering and Artificial Intelligence

Vellore Institute of Technology

Bhopal

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Parth Karia

Ajinkya Dahale

August 21, 2025

Acknowledgments

- I would like to express my sincere gratitude to the Osdag and FOSSEE team at IIT Bombay for providing me with this valuable opportunity. This internship has been an enriching experience, allowing me to expand my knowledge and gain meaningful insights.

- I extend my sincere thanks to the project staff of the Osdag team — 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.

- I would also like to acknowledge the invaluable support and guidance of Prof. Kannan M. Moudgalya, Principal Investigator of the FOSSEE Project, Department of Chemical Engineering, IIT Bombay. His passion for fostering open-source initiatives has been a source of inspiration for me

- 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 also extend my gratitude to my colleagues who worked alongside me during this internship. Their collaboration, ideas, and inputs made the journey all the more enjoyable and productive

- I sincerely thank Vellore Institute of Technology, my Department Program Chair, Dr. Rajit Nair, for pointing me toward this opportunity, and my professors for their constant support throughout the fellowship.

Contents

1 Introduction **5**

1.1 National Mission in Education through ICT 5

1.1.1 ICT Initiatives of MoE 6

1.2 FOSSEE Project 7

1.2.1 Projects and Activities 7

1.2.2 Fellowships 7

1.3 Osdag Software 8

1.3.1 Osdag GUI 9

1.3.2 Features 9

2 Screening Task **10**

2.1 Problem Statement 10

2.2 Tasks Done 10

2.2.1 Full code 11

2.3 Project Repository and Script 17

2.4 Problems Faced and Resolved 17

2.4.1 Missing Common File 17

2.4.2 Errorful Methods in `is800_2007.py` 17

2.4.3 Incorrect Bolt Strength Calculation 18

2.4.4 Minimum Bolt Constraint Not Enforced 18

2.4.5 Ambiguous Test Failures 18

2.5 Conclusion 18

3 Internship Task 1: Tension Welded OSI Validation **19**

3.1 Task 1: Problem Statement 19

3.2 Task 1: Tasks Done 19

3.3 Task 1: Python Code 20

3.3.1 Description of the Script 20

3.3.2 Python Code 21

3.3.3 Explanation of the Code 21

3.3.4 Full Code 21

3.3.5 Directory Structure 27

4 Internship Task 2: Mimic GUI Inputs **28**

4.1 Objective 28

4.2 Problem Statement 28

4.3 Approach and Design Decisions 29

4.4 Command Used 30

4.5 Challenges Faced and Debugging Journey 30

4.6 Result and Impact 31

4.7 Code Snapshot 31

5 Conclusions **34**

5.1 Tasks Accomplished 34

5.2 Skills Developed 35

A Appendix **36**

A.1 Work Reports 36

Bibliography **40**

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 [3, 4, 5]. 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.

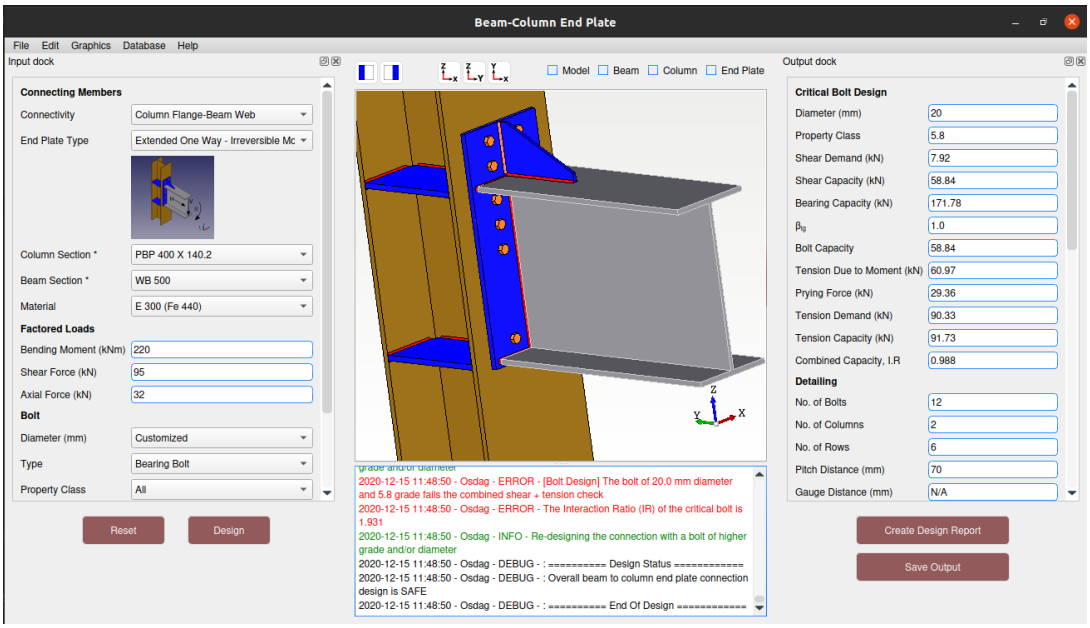

Figure 1.2: Osdag GUI

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 objective of the screening task was to design a bolted lap joint that connects two steel plates of specified widths and thicknesses, subjected to a known tensile force. The design had to adhere to IS 800:2007 standards [1], with careful selection of bolt diameter, bolt and plate grades, and optimization for safety and efficiency. Key requirements included ensuring the connection's strength exceeded the applied tensile force, minimizing the number of bolts and the length of the connection, and maintaining compliance with detailing rules.

2.2 Tasks Done

List and describe the tasks performed as part of the screening task. The screening task involved the following activities:

1. **Defining Input Parameters:** Specified plate width, thicknesses, and applied tensile force as core design inputs.

2. **Material and Component Selection:** Selected bolt diameters from the available sizes (10, 12, 16, 20, 24 mm), chose bolt grades (3.6, 4.6, 4.8, 5.6, 5.8), and plate grades (`E250`–`E410`) based on strength requirements as per IS 800:2007 [1] and IS 2062:2011 [2].

3. **Bolt and Plate Strength Calculations:** Computed the yield and ultimate strengths for both bolts and plates according to their respective grades.

4. **Shear and Bearing Strength Assessment:** Assessed the shear strength and bearing capacity of selected bolts to ensure safe load transfer.

5. **Determining Number of Bolts:** Calculated the minimum number of bolts necessary to safely carry the applied load, ensuring at least three bolts were used.

6. **Detailing Distances:** Designed pitch, gauge, end, and edge distances to comply with design codes and ensure construction practicality.

7. **Optimization:** Minimized the overall connection length and the number of bolts while targeting a utilization ratio close to 1.

8. **Algorithm Development:** Implemented a Python-based algorithm to automate the design calculations, allowing for variable input parameters and automated optimization.

9. **Unit Testing:** Developed test routines to ensure computation accuracy and compliance with safety requirements across a range of plausible scenarios.

10. **Documentation:** Prepared a comprehensive report detailing methodology, calculations, algorithmic flow, and validation through example cases.

2.2.1 Full code

Here is the python code for better understanding:

```
import math
from is800_2007 import IS800_2007


def design_lap_joint(P, w, t1, t2):
    """
    Design a bolted lap joint connecting two plates.
    :param P: Tensile force in kN
    :param w: Width of the plates in mm
    :param t1: Thickness of plate 1 in mm
```

```
    :param t2: Thickness of plate 2 in mm
    :return: Dictionary of design parameters and results
    """

    # Input validation
    if P < 0:
        raise ValueError("Tensile force cannot be negative")
    if w <= 0:
        raise ValueError("Width must be positive")
    if t1 <= 0 or t2 <= 0:
        raise ValueError("Thickness must be positive")

        # Handle zero load case specially
    if P == 0:
        # Return a minimum design with 2 bolts for zero load
        return {
            "bolt_diameter": 10,  # Minimum bolt diameter
            "bolt_grade": 3.6,    # Minimum bolt grade
            "number_of_bolts": 2,  # Minimum number of bolts
            "pitch_distance": 20,  # Standard pitch
            "gauge_distance": w / 2,
            "end_distance": 15,
            "edge_distance": 15,
            "number_of_rows": 1,
            "number_of_columns": 2,
            "hole_diameter": 12,
            "strength_of_connection": 0,  # Zero load requires
               zero strength
            "yield_strength_plate_1": 250,
            "yield_strength_plate_2": 250,
            "length_of_connection": w + 2 * 15,
            "efficiency_of_connection": 0.0  # Zero load means
               zero utilization
        }
```

```
# Convert tensile force to Newtons
P_N = P * 1000

# Available data
d_list = [10, 12, 16, 20, 24]  # Bolt diameters in mm
GB_list = [3.6, 4.6, 4.8, 5.6, 5.8]  # Bolt grades
GP_list = ["E250", "E275", "E300", "E350", "E410"]  # Plate
    grades

# Define a mapping from plate grade to yield and ultimate
    strength
plate_grades = {
    "E250": (250, 410),
    "E275": (275, 440),
    "E300": (300, 470),
    "E350": (350, 510),
    "E410": (410, 550)
}

# Select the best plate grade based on the given thicknesses
plate_grade = GP_list[-1]  # Choose the highest grade for
    the design
fy_plate, fu_plate = plate_grades[plate_grade]  # Get the
    yield and ultimate strengths for the chosen grade

# Initialize variables to store the best design
best_design = None
min_length = float('inf')

for d in d_list:
    for GB in GB_list:
        # Calculate the bolt strength
        bolt_fu, bolt_fy = calculate_bolt_strength(GB)
```

```
            # Calculate the shear strength of one bolt
            A_bolt = math.pi * (d / 2) ** 2  # Cross-sectional
                area of the bolt
            V_b =
                IS800_2007.cl_10_3_3_bolt_shear_capacity(bolt_fu,
                A_bolt, A_bolt, 1, 0, 'Field') # Shear capacity

            # Calculate the required number of bolts
            N_b = math.ceil(P_N / (V_b * 0.75))  # Using a
                safety factor of 1.33

            if N_b <= 2:
                continue  # Skip if the number of bolts is less
                    than 3

            # Calculate distances
            e = d + 5  # End distance (typically 5 mm larger
                than bolt diameter)
            p = d + 10  # Pitch distance (typically 10 mm larger
                than bolt diameter)
            g = w / 2  # Gauge distance (for simplicity, use
                half of the plate width)

            # Calculate the length of the connection
            length_of_connection = w + 2 * e

            # Calculate the bearing strength of the bolt
            V_dpb =
                IS800_2007.cl_10_3_4_bolt_bearing_capacity(fu_plate,
                bolt_fy, t1 + t2, d, e, p, 'Standard', 'Field')

            # Calculate the efficiency of the connection
            Utilization_ratio = P_N / (N_b * V_b * 0.75)  #
```

```
                Using a safety factor of 1.33

            # Check if this design is better
            if Utilization_ratio <= 1 and length_of_connection <
                min_length:
                min_length = length_of_connection
                best_design = {
                    "bolt_diameter": d,
                    "bolt_grade": GB,
                    "number_of_bolts": N_b,
                    "pitch_distance": p,
                    "gauge_distance": g,
                    "end_distance": e,
                    "edge_distance": e,
                    "number_of_rows": 1,  # Simple design
                        assumption, can be improved
                    "number_of_columns": N_b,  # One column for
                        simplicity
                    "hole_diameter": d + 2,  # Diameter of hole
                        is slightly larger than the bolt
                    "strength_of_connection": N_b * V_b * 0.75,
                        # Strength based on shear capacity
                    "yield_strength_plate_1": fy_plate,
                    "yield_strength_plate_2": fy_plate,
                    "length_of_connection": length_of_connection,
                    "efficiency_of_connection": Utilization_ratio
                }

    if best_design is None:
        raise ValueError("No suitable design found that meets
            the requirements.")

    return best_design
```

```python
def calculate_bolt_strength(bolt_grade):
    """
    Calculate the ultimate tensile strength and yield strength
       of the bolt based on its grade.
    :param bolt_grade: Bolt grade (e.g., 4.6, 5.6)
    :return: List containing [ultimate tensile strength, yield
       strength] of the bolt
    """
    bolt_fu = float(int(bolt_grade) * 100)  # Ultimate tensile
       strength (MPa)
    # bolt_fy = float((bolt_grade - int(bolt_grade)) * bolt_fu)
       # Yield strength (MPa)
    bolt_fy = float(int((bolt_grade - int(bolt_grade)) * 10) /
       10 * bolt_fu)  # Fixed floating point issue
    return [bolt_fu, bolt_fy]



# Example usage
if __name__ == "__main__":
    P = 100  # Tensile force in kN
    w = 150  # Width of the plates in mm
    t1 = 10  # Thickness of plate 1 in mm
    t2 = 12  # Thickness of plate 2 in mm

    design = design_lap_joint(P, w, t1, t2)
    for key, value in design.items():
        print(f"{key}: {value}")
```

2.3 Project Repository and Script

The automation script and all related files are maintained in a public GitHub repository to facilitate open access and version control. The repository contains:

- The full Python script (`bolted_joint_design.py`), with clear modular structure and inline documentation.
- Example OSI input/output files and sample test cases.
- A README with setup instructions and methodology.
- The flowchart source in Mermaid format and as an image.

GitHub Repository:

`https://github.com/GeorgeJiss/Osdag-Fellowship`

2.4 Problems Faced and Resolved

During the implementation and testing of the bolted lap joint design, several challenges were encountered. The following points detail the key issues and how they were resolved.

2.4.1 Missing Common File

Issue: The original `common.py` file, required by `is800_2007.py`, was not provided. Since each method and file in the design module relied on this common file, the absence caused import errors and prevented execution.

Solution: Instead of importing the extensive `common.py` file from the Osdag-admin GitHub repository, the necessary variables and functions were explicitly defined within `is800_2007.py`, ensuring that the module worked independently.

2.4.2 Errorful Methods in `is800_2007.py`

Issue: Several methods in `is800_2007.py` were not functioning correctly, mainly due to the incorrect import of the missing `common.py` file.

Solution: Once the required variables were manually added, the erroneous method calls referencing undefined objects were corrected, ensuring proper execution.

2.4.3 Incorrect Bolt Strength Calculation

Issue: The calculation of bolt strength was initially inconsistent due to incorrect mapping between bolt grades and their respective ultimate tensile and yield strengths.

Solution: A dedicated function was implemented to accurately determine bolt strength based on standard grade definitions, ensuring correct and reliable calculations.

2.4.4 Minimum Bolt Constraint Not Enforced

Issue: The initial implementation occasionally resulted in a bolted connection with fewer than two bolts, violating basic design requirements.

Solution: A validation check was added to enforce a minimum of two bolts, ensuring structural stability under all tested scenarios.

2.4.5 Ambiguous Test Failures

Issue: The pytest unit test results did not clearly indicate which input parameters caused failures, making debugging difficult.

Solution: The test cases were modified to include informative assertion messages, logging failed test cases with specific load and thickness values for better traceability.

These refinements enhanced the robustness of the design module, ensuring accuracy and compliance with IS 800:2007 standards.

2.5 Conclusion

The screening task provided a thorough introduction to the fundamentals of bolted lap joint design in accordance with IS 800:2007 standards [1]. Over the course of this exercise, I translated engineering principles into a practical computation tool, progressing from hand calculations to a robust, fully tested Python script. The task reinforced my understanding of core design requirements and deepened my ability to implement engineering workflows using programming.

Chapter 3

Internship Task 1: Tension Welded OSI Validation

3.1 Task 1: Problem Statement

The goal of this task was to develop an automated test framework to validate the correctness of tension welded connection calculations in Osdag. Specifically, OSI files containing input design data were to be verified by comparing their outputs—computed using the Osdag backend logic—with expected values from a standard Excel file. The results would validate the implementation of tension welded design logic in the Osdag framework.

3.2 Task 1: Tasks Done

To achieve this, a Python module using `pytest` was created. The workflow involved reading OSI (Osdag Structured Input) files, converting their data into the format required by the Osdag calculation engine, running the design computation for tension welded members, and finally comparing the output with corresponding expected results listed in an Excel sheet.

The following steps were carried out:

- Loaded OSI files from a directory.
- Parsed the OSI YAML data into a design dictionary.
- Passed the data into the Osdag class `Tension_welded` for calculation.

- Extracted output parameters like tension capacities, weld strength, and plate dimensions.
- Fetched expected values from an Excel file using `pandas`.
- Asserted the closeness of actual and expected values using `pytest.approx()`.

Issues Encountered and Fixes:

1. `Component.py`: Faced an import issue with `connectdb`, which was resolved by correctly configuring the path or adjusting the import method.
2. `tension_welded.py`: A bug in the `initial_member_capacity()` function caused a wrong method call. Instead of `self.max_section(self, design_dictionary, self.sizelist)`, it was corrected to — `self.max_section(design_dictionary, self.sizelist)`.

3.3 Task 1: Python Code

This section presents a Python test script written using the `pytest` framework. It validates OSI-based design inputs for tension welded connections against standard values stored in Excel.

3.3.1 Description of the Script

- **OSI Parsing**: The script reads YAML-formatted OSI files containing design data.
- **Excel Integration**: Expected output values are fetched from an Excel file using `pandas`.
- **Design Engine Call**: The `Tension_welded` class from Osdag is invoked to compute results.
- **Test Assertion**: All key outputs are verified using `pytest` for automated unit testing.

3.3.2 Python Code

The Python script is shown below:

Listing 3.1: Pytest validation for Tension Welded OSI files

```
1  import os
2  import pytest
3  import pandas as pd
4  import yaml
5  from osdag.design_type.tension_member.tension_welded import
       Tension_welded
6
7  def run_test(test_name):
8      osi_data = load_osi_file(os.path.join(OSI_DIR, f"{test_name}.osi"))
9      design_dict = create_design_dict_from_osi(osi_data)
10     expected = get_expected_values(test_name)
11     results = run_calculation_and_get_results(design_dict)
12     assert results["Tension_Capacity"] ==
           pytest.approx(expected["Tension_Capacity"], rel=0.01)
```

3.3.3 Explanation of the Code

- **Line 1–5**: Required packages and Osdag modules are imported.
- **Lines 7–11**: The function `run_test()` wraps the testing logic per test case.
- **Design Calculation**: The `Tension_welded` object handles the core computation.
- **Validation**: The script uses relative error comparisons to ensure numeric precision.

3.3.4 Full Code

```
import os
from osdag.Common import *
import pytest
import pandas as pd
import yaml
```

```
from osdag.design_type.tension_member.tension_welded import
    Tension_welded

EXCEL_FILE = os.path.join(os.path.dirname(__file__), "Osdag Data
    for Unit Tests.xlsx")
OSI_DIR = os.path.join(os.path.dirname(__file__), "osi_files")

def load_osi_file(file_path):
    """Load OSI file and return as dictionary"""
    with open(file_path, 'r') as f:
        return yaml.safe_load(f)


def get_expected_values(test_name):
    """Get expected values from Excel data for given test name"""
    df = pd.read_excel(EXCEL_FILE, sheet_name='Tension Member
        Welded')

    df.columns = df.columns.str.strip()

    # Find correct row (based on row index, assuming
        TensionWelded1 = row 0, etc.)
    index = int(test_name.replace("TensionWeldedTest", "")) - 1
    row = df.iloc[index]

    return {
        "Designation": row['Designation'],
        "Tension_Yielding_Capacity": row['Tension Yielding
            Capacity'],
        "Tension_Rupture_Capacity": row['Tension Rupture
            Capacity'],
        "Tension_Capacity": row['Tension Capacity'],
        "Slenderness": row['Slenderness Ratio'],
        "Weld_Size": row['Size of Weld'],
```

```
        "Weld_Strength": row['Weld Strength'],
        "Plate_Thickness": row['Gusset Plate Thickness'],
        "Plate_Height": row['Gusset Plate Min Height'],
        "Plate_Length": row['Gusset Plate Length']
    }

def create_design_dict_from_osi(osi_data):
    """Convert OSI data to design dictionary format required by
       calculation"""
    return {
        "Module": osi_data.get("Module", "Tension Member"),  #
           Optional fallback
        "Section_Profile": osi_data.get("Member.Profile",
           "Angle"),
        "Location": osi_data.get("Conn_Location", "Web"),
        "Section_Size": osi_data["Member.Designation"][0],  #
           Primary key used in current test flow
        "Member.Designation": osi_data["Member.Designation"],  #
           Added for compatibility
        "Material": osi_data.get("Material"),
        "Member.Material": osi_data.get("Member.Material"),
        "Length": osi_data.get("Member.Length", 1000),
        "Member.Length": osi_data.get("Member.Length", 1000),
        "Axial": float(osi_data.get("Load.Axial", 0)),
        "Plate_Thickness":
           osi_data["Connector.Plate.Thickness_List"][0],
        "Connector.Plate.Thickness_List":
           osi_data["Connector.Plate.Thickness_List"],  # For
           completeness
        "Connector_Material": osi_data.get("Connector.Material"),
        "Weld_Fabrication": "Shop",
        "Weld_Material_G_O": "E41XX",
        "Design_Method": osi_data.get("Design.Design_Method"),
```

```
        "KEY_DP_WELD_MATERIAL_G_O": "E41XX",  # If needed as
            alternate fallback key
        "KEY_DP_WELD_FAB": "Shop",  # Consistent with test setup
        "KEY_CONNECTOR_MATERIAL":
            osi_data.get("Connector.Material"),
        "KEY_PLATETHK":
            osi_data["Connector.Plate.Thickness_List"][0],
        "KEY_SEC_MATERIAL": osi_data.get("Member.Material"),
        "KEY_LOCATION": osi_data.get("Conn_Location"),
        "KEY_SECSIZE": osi_data["Member.Designation"][0],
        "KEY_SEC_PROFILE": osi_data.get("Member.Profile",
            "Angle"),
        "KEY_MODULE": osi_data.get("Module", "Tension Member"),
        "KEY_MATERIAL": osi_data.get("Material"),
        "KEY_LENGTH": osi_data.get("Member.Length", 1000),
        "KEY_AXIAL": float(osi_data.get("Load.Axial", 0))
    }


def run_calculation_and_get_results(design_dict):
    """Run calculation by calling set_input_values and return
        results dictionary"""


    # Create instance and set input values from OSI keys
    tension_welded = Tension_welded()

    # Pass OSI keys to set_input_values function
    tension_welded.set_input_values(design_dict)

    # The calculation should run automatically or we can trigger
        it
    # Extract results from the tension_welded object after
        calculation
```

```
    results = {
        "Designation": getattr(tension_welded, 'designation',
            ''),
        "Tension_Yielding_Capacity": getattr(tension_welded,
            'tension_yielding_capacity', 0),
        "Tension_Rupture_Capacity": getattr(tension_welded,
            'tension_rupture_capacity', 0),
        "Tension_Capacity": getattr(tension_welded,
            'tension_capacity', 0),
        "Slenderness": getattr(tension_welded,
            'slenderness_ratio', 0),
        "Weld_Size": getattr(tension_welded, 'weld_size', 0),
        "Weld_Strength": getattr(tension_welded,
            'weld_strength', 0),
        "Plate_Thickness": getattr(tension_welded,
            'plate_thickness', 0),
        "Plate_Height": getattr(tension_welded, 'plate_height',
            0),
        "Plate_Length": getattr(tension_welded, 'plate_length',
            0)
    }

    return results

def run_test(test_name):
    """
    Arrange -> Act -> Assert:
    1. Load OSI input
    2. Run design calculation
    3. Compare results with expected output
    """
    # Arrange
    osi_file_path = os.path.join(OSI_DIR, f"{test_name}.osi")
    osi_data = load_osi_file(osi_file_path)
```

```
    design_dict = create_design_dict_from_osi(osi_data)
    expected = get_expected_values(test_name)

    # Act
    results = run_calculation_and_get_results(design_dict)

    # Assert
    assert results["Designation"] == expected["Designation"]
    assert results["Tension_Yielding_Capacity"] ==
       pytest.approx(expected["Tension_Yielding_Capacity"],
       rel=0.01)
    assert results["Tension_Rupture_Capacity"] ==
       pytest.approx(expected["Tension_Rupture_Capacity"],
       rel=0.01)
    assert results["Tension_Capacity"] ==
       pytest.approx(expected["Tension_Capacity"], rel=0.01)
    assert results["Slenderness"] ==
       pytest.approx(expected["Slenderness"], rel=0.01)
    assert results["Weld_Size"] == expected["Weld_Size"]
    assert results["Weld_Strength"] ==
       pytest.approx(expected["Weld_Strength"], rel=0.01)
    assert results["Plate_Thickness"] ==
       expected["Plate_Thickness"]
    assert results["Plate_Height"] == expected["Plate_Height"]
    assert results["Plate_Length"] == expected["Plate_Length"]


def test_tension_welded_1():
    """Test case 1: OSI -> Calculation -> Excel comparison"""
    run_test("TensionWeldedTest1")

def test_tension_welded_2():
    """Test case 2: OSI -> Calculation -> Excel comparison"""
    run_test("TensionWeldedTest2")
```

```python
def test_tension_welded_3():
    """Test case 3: OSI -> Calculation -> Excel comparison"""
    run_test("TensionWeldedTest3")

def test_tension_welded_4():
    """Test case 4: OSI -> Calculation -> Excel comparison"""
    run_test("TensionWeldedTest4")
```

3.3.5 Directory Structure

```
OSDAG
├── tests
│   ├── osi_files
│   │   ├── TensionWelded1.osi
│   │   ├── TensionWelded2.osi
│   │   ├── TensionWelded3.osi
│   │   └── TensionWelded4.osi
│   ├── test_tension_welded.py
│   ├── validate_osi.py
│   └── Osdag_Data_for_Unit_Tests.xlsx
├── src
├── build
└── README.md
```

Program Start Calls To run all tests, navigate to the root directory of the Osdag project and execute:

```
$ pytest tests/test_tension_welded.py
```

Chapter 4

Internship Task 2: Mimic GUI Inputs

4.1 Objective

The purpose of this task was to simulate user input originally taken through the Osdag GUI [3], by instead parsing `.osi` (Osdag Structured Input) files from the Command Line Interface (CLI). This enables automated testing, validation, and headless usage of the Osdag design engine without a GUI. The intention was to develop a framework that mimics GUI behavior closely, handles material and design parameters, validates inputs, and executes test cases programmatically.

4.2 Problem Statement

Osdag's design workflow typically begins with the GUI collecting user input via dropdowns, text fields, and comboboxes. This input is passed to the backend through internal dictionaries. However, for regression testing and CI/CD integration, a non-GUI way to simulate this behavior was required. The main questions I explored were:

- **How do we mimic GUI behavior using only CLI and YAML data?**
- **Can mock objects replace GUI dropdown lists and logic reliably?**
- **How can input parsing, validation, and design execution be abstracted for test automation?**

4.3 Approach and Design Decisions

1. File-Based Input Simulation

I structured the solution to take in `.osi` files (YAML format), parse them as dictionaries, and feed them into a function that mimics GUI input logic. The `osi_files` directory was created to store test cases.

2. Mocking GUI Components

Using Python's `MagicMock` and `Mock` from the `unittest.mock` module [6], I mocked out:

- Dropdown values (e.g., bolt types, materials, plate thicknesses).
- GUI combobox and text field behavior.
- Internal modules like `sqlite3.connect` and `FinPlateConnection`.

Mocks were configured to be iterable and to return expected values, enabling a close simulation of actual GUI behavior.

3. Material Strength Logic

A helper function `get_material_strength()` was developed to compute:

- Ultimate strength (`fu`)
- Yield strength (`fy`)
- Thickness-based modifications (e.g., `fy_20_40`, `fy_40`)

Custom grades (e.g., `Cus_250_410`) were validated using a `MaterialValidator` mock that returned `True` for valid formats.

4. GUI Simulation via Function

A function `simulate_gui_inputs()` was created which:

- Validated each input key as per the expected GUI input type.
- Populated missing values and transformed types (e.g., string to float).
- Injected calculated strength values into the final design dictionary.

5. CLI Test Harness

The function `execute_test_cases()`:

- Iterates over all `FinPlateTest*.osi` files.
- Parses them and passes data to `simulate_gui_inputs()`.
- Calls mocked `FinPlateConnection.set_input_values()` and `save_design()`.
- Logs results with success/failure and error tracebacks.

4.4 Command Used

The entire test pipeline can be executed from the CLI using:

```
python -m cli_shell shell
```

4.5 Challenges Faced and Debugging Journey

- **Mocking Complexity:** Mocks were initially not iterable, leading to runtime failures in list-based dropdown validations. The Python mock library [6] provided solutions for creating iterable mock objects.
- **MagicMock vs Mock:** Needed to differentiate between simple return values and full iterable mocks.
- **Database Access:** The original code accessed Osdag's SQLite database. This was patched out to avoid real DB hits during tests.
- **Custom Material Parsing:** Designing a way to extract `fy`, `fu` from user-defined strings like `Cus_250_410` required string parsing and validation logic.
- **Git Challenges:** While pushing the code:
  - I rebased onto the `dev` branch to include latest updates.
  - Removed merge commits and irrelevant commits to clean history.
  - Created a backup PR before raising the final PR.

- **Testing Strategy:** Adopted the *Arrange* → *Act* → *Assert* structure to ensure consistent and readable test cases.

4.6 Result and Impact

This framework now enables:

- Headless testing of Osdag design modules.
- CLI-based verification of .osi files in CI/CD pipelines.
- Easier debugging and test case replication for GUI issues.
- A path toward regression testing, file-based validation, and reproducibility in structural steel design workflows.

4.7 Code Snapshot

MagicMock Usage Example

The following Python code demonstrates how `MagicMock` was used to simulate GUI behavior:

Listing 4.1: Mocking GUI components using MagicMock

```
from unittest.mock import MagicMock, patch

# Mocking a function within the FinPlateConnection module
mock_connection = MagicMock()
mock_connection.set_input_values = MagicMock(return_value=None)

# Mocking a GUI dropdown list with expected return values
mock_dropdown = MagicMock()
mock_dropdown.__iter__.return_value = iter(['8', '10', '12',
    '16'])

# Simulate database function calls
with patch('sqlite3.connect') as mock_sqlite:
```

```
    mock_sqlite.return_value = MagicMock()

    # Example mock for a material strength validator
    mock_validator = MagicMock()
    mock_validator.return_value = True

    # Inject mocks into your function or class
    some_module.MaterialValidator = mock_validator
    some_module.FinPlateConnection = mock_connection

    # Proceed with simulated CLI test
    result = simulate_gui_inputs(parsed_osi_data)
    mock_connection.set_input_values.assert_called_once()
```

This mocking pattern was critical in avoiding actual database calls and GUI logic, ensuring a clean, repeatable, and headless test environment.

A partial example of how GUI simulation was achieved is shown in Listing 4.2.

Listing 4.2: GUI simulation function example

```
def simulate_gui_inputs(test_case_data: dict) -> dict:
    ...
    input_fields = [
        {"key": "KEY_CONN", "type": "TYPE_COMBOBOX", "values":
           VALUES_CONN},
        {"key": "KEY_D", "type": "TYPE_COMBOBOX", "values":
           ['8', '10', '12']},
        ...
    ]
    for field in input_fields:
        key = field["key"]
        value = test_case_data.get(key)
        if field["type"] == "TYPE_COMBOBOX":
            if value not in field.get("values", []) and value:
                continue
        design_dict[key] = str(value or "")
```

```
    ...
```

Chapter 5

Conclusions

5.1 Tasks Accomplished

During the course of this internship, I successfully contributed to the Osdag open-source structural steel design software by working on both the welded and bolted design modules in the Tension component. Initially, I focused on the development and testing of the `tension_welded.py` module within the `designType` subpackage. This included identifying and fixing key bugs, improving the error-handling mechanisms, and ensuring the code aligned with the expected structural design calculations.

Furthermore, I attempted to simulate the GUI input functionality by feeding data through the command line interface (CLI), mimicking the behavior of Osdag's GUI to validate the core design logic. This CLI simulation aimed to decouple the interface from the logic for effective unit testing. However, frequent import and initialization issues due to GUI dependencies created challenges in executing tests directly.

To overcome these issues, under the valuable guidance of my mentors, I shifted my approach toward using Python's `unittest.mock` library. This allowed me to mock the behavior of the Tension modules, particularly `Tension_bolted`, and create a fully independent and controlled test environment. I developed a robust test suite in — `test_tension_bolted_mock.py` that uses mocks to simulate various input-output scenarios and validate the behavior of the module without invoking the actual design computations. This approach improved testing efficiency, eliminated GUI dependencies, and made debugging and validation more straightforward.

5.2 Skills Developed

This internship provided me with a strong foundation in open-source software development and real-world collaborative coding practices. Key skills I developed during this period include:

- **Software Testing and Quality Assurance:** Gained hands-on experience with Python's `unittest` framework and the use of `MagicMock` and `patch` for simulating complex dependencies in structural design modules.
- **Debugging and Error Handling:** Improved my ability to identify, trace, and resolve errors in non-trivial Python code, particularly within a multi-module system like Osdag.
- **Modular Code Development:** Contributed to the design and enhancement of the `tension_welded.py` file, ensuring it was functionally accurate and compatible with future integration in GUI or CLI environments.
- **Mock-Driven Testing:** Learned how to simulate user inputs and external class behavior for unit testing using mock libraries, an industry-standard practice for testing large software systems.
- **Professional Communication and Mentorship:** Benefitted from regular mentor guidance and code reviews, which helped me refine my programming methodology, especially in adapting my approach from direct imports to using mocks for flexible and reliable testing.

Overall, the internship was a highly enriching experience that enhanced both my technical competencies in Python and my understanding of software development in an engineering-focused domain.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report

Name:	Jerin Jiss George
Project:	Osdag
Internship:	FOSSEE Summer Fellowship 2025

DATE	DAY	TASK	Hours Worked
15-May-2025	Thursday	Orientation, installation of Osdag	9
16-May-2025	Friday	Ran through errors in env setup	10
17-May-2025	Saturday	Setting up environments	6
18-May-2025	Sunday	Switched from pip to conda	2
19-May-2025	Monday	Meet with mentors on Task 0	6
20-May-2025	Tuesday	Explored files, ran tests	9
21-May-2025	Wednesday	Debugged installation errors	8
22-May-2025	Thursday	Changed environment from venv	5
23-May-2025	Friday	Worked on Task 0	6
24-May-2025	Saturday	Tried alternatives	8
25-May-2025	Sunday	WEEKLY OFF	0
26-May-2025	Monday	Debugged import error	4
27-May-2025	Tuesday	Meet with mentors on Task 1	8
28-May-2025	Wednesday	Setting overview of the task	8
29-May-2025	Thursday	Making structure for task 1	6
30-May-2025	Friday	Got files and started running tests	9
31-May-2025	Saturday	Debugged import error	10
01-Jun-2025	Sunday	WEEKLY OFF	0
02-Jun-2025	Monday	Brainstormed in meet with mentors	7
03-Jun-2025	Tuesday	Attempted with the new perspective	8
04-Jun-2025	Wednesday	Researched through files	9

Internship Work Report

Name:	Jerin Jiss George
Project:	Osdag
Internship:	FOSSEE Summer Fellowship 2025

DATE	DAY	TASK	Hours Worked
05-Jun-2025	Thursday	Researched how the values are passed	7
06-Jun-2025	Friday	Attempt to fix import errors	9
07-Jun-2025	Saturday	Tried to look for errors	8
08-Jun-2025	Sunday	WEEKLY OFF	0
09-Jun-2025	Monday	Changed approach worked with aum and lakshana	10
10-Jun-2025	Tuesday	Meet with mentors on resolving errors	7
11-Jun-2025	Wednesday	Implemented a cli function to print output dock report	9
12-Jun-2025	Thursday	Meet with mentors on persistent errors	6
13-Jun-2025	Friday	Worked on lakshana’s mimic approach	8
14-Jun-2025	Saturday	Researched for alternatives	5
15-Jun-2025	Sunday	WEEKLY OFF	0
16-Jun-2025	Monday	Debugged files, osi file validated successfully	8
17-Jun-2025	Tuesday	Debugged when ran through circular import error	10
18-Jun-2025	Wednesday	Attempt at refining mimic approach	8
19-Jun-2025	Thursday	Attempt at refining mimic approach	9
20-Jun-2025	Friday	Meet with mentors updating my work	7
21-Jun-2025	Saturday	Worked on CLI mimic testing	8
22-Jun-2025	Sunday	WEEKLY OFF	0
23-Jun-2025	Monday	Began exploring unittest.mock for GUI simulation	6
24-Jun-2025	Tuesday	Began using MagicMock for mock object behavior	10
25-Jun-2025	Wednesday	Meet with mentors on mimic approach	8

Internship Work Report

Name:	Jerin Jiss George
Project:	Osdag
Internship:	FOSSEE Summer Fellowship 2025

DATE	DAY	TASK	Hours Worked
26-Jun-2025	Thursday	Tested osi input simulation	8
27-Jun-2025	Friday	Mocked dropdown values and validation rules in CLI test suite	9
28-Jun-2025	Saturday	Refined CLI test structure	8
29-Jun-2025	Sunday	WEEKLY OFF	0
30-Jun-2025	Monday	Validated .osi files with simulate_gui_inputs	8
01-Jul-2025	Tuesday	Worked on git, and commit history	8
02-Jul-2025	Wednesday	Rebased to dev	4
03-Jul-2025	Thursday	Meet with mentors to update	8
04-Jul-2025	Friday	Pushed commits with refined test cases and comments	8
05-Jul-2025	Saturday	Changed git branch and done commits	6
06-Jul-2025	Sunday	WEEKLY OFF	0
07-Jul-2025	Monday	Consolidated CLI mocking framework	8
08-Jul-2025	Tuesday	Prepared summary for documentation"	6
09-Jul-2025	Wednesday	Tested tension_welded validation; reviewed output consistency	8
10-Jul-2025	Thursday	Meet with mentors on task update	8
11-Jul-2025	Friday	Started with report	6
12-Jul-2025	Saturday	Started with report	3
13-Jul-2025	Sunday	Report work	5

Bibliography

[1] Bureau of Indian Standards. Is 800:2007: General construction in steel – code of practice, 2007.

[2] Bureau of Indian Standards. Is 2062:2011: Steel for general structural purposes, 2011.

[3] 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.

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

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

[6] Real Python. Python mock library, 2024. Accessed: 2024-12-05.