FOSSEE Summer Internship Report

On

Development, Analysis, and Quality Assurance of Steel Connection Modules for the Osdag Framework

Submitted by

Nishi Kant Mandal

3rd Year B.Tech Student, Department of Civil Engineering

B.I.T. Sindri

Sindri, Dhanbad, Jharkhand

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

July 26, 2025

Acknowledgments

I would like to extend my sincere gratitude to the following individuals and organizations whose guidance and support were instrumental in the successful completion of this fellowship project:

- My heartfelt thanks go to the Osdag team, particularly **Ajmal Babu M. S.**, **Ajinkya Dahale**, and **Parth Karia**, for their continuous technical expertise and collaborative support during the development of the Welded Butt and Lap Joint modules.
- I am truly grateful for the mentorship and inspiring leadership of **Prof. Siddhartha Ghosh**, Principal Investigator of the Osdag project, from the Department of Civil Engineering at IIT Bombay.
- I would like to express my gratitude to **Prof. Kannan M. Moudgalya**, Principal Investigator of the FOSSEE project, and the entire FOSSEE team in the Department of Chemical Engineering at IIT Bombay for their invaluable project guidance and academic support.
- My sincere appreciation is extended to **Usha Viswanathan**, **Vineeta Parmar**, and the FOSSEE management team for their exceptional administrative support, which ensured the seamless progression of this internship.
- I gratefully acknowledge the financial and institutional support from the **National Mission on Education through Information and Communication Technology (NMEICT)**, Ministry of Education, Government of India.
- I wish to express my sincere appreciation to my colleague, **Tanu Singh**. Her commitment and collaborative spirit significantly eased the challenges of module development and made this fellowship a both fruitful and enjoyable experience.

- Finally, I would like to thank my alma mater, **B.I.T. Sindri**, along with the mentors and administrative staff who have supported me throughout my academic journey.

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 Internship Task 1: Development of Design and Detailing Checklists for Welded Connections **13**

2.1 Task 1: Problem Statement 13

2.2 Task 1: Tasks Done 14

2.2.1 Methodology and Implementation 14

2.2.2 Development of DDCL for Welded Lap Joint 14

2.2.3 Development of DDCL for Welded Butt Joint 16

2.3 Task 1: Documentation 18

2.3.1 Purpose of the DDCL in Osdag 18

3 Internship Task 2: Implementation of the Welded Butt Joint Design Module **19**

3.1 Task 2: Problem Statement 19

3.2 Task 2: Tasks Done 20

3.2.1 Module Architecture and Integration 20

3.2.2 Core Design Logic Implementation 20

3.2.3 Output and Reporting 21

3.3 Task 2: Python Code 22

3.3.1 Description of the Script 22

3.3.2 Code Snippets and Explanation 22

3.3.3 Full code 25

3.4 Task 2: Documentation 66

4 Internship Task 3: Implementation of the Welded Lap Joint Design Module 68

4.1 Task 3: Problem Statement 68
4.2 Task 3: Tasks Done 68
4.2.1 Module Architecture and Integration 69
4.2.2 Core Design Logic Implementation 69
4.2.3 Output and Reporting 70
4.3 Task 3: Python Code 71
4.3.1 Description of the Script 71
4.3.2 Code Snippets and Explanation 71
4.3.3 Full code 73
4.4 Task 3: Documentation 95

5 Internship Task 4: Bug Detection and Quality Assurance for Splice Connection Modules 96

5.1 Task 4: Problem Statement 96
5.2 Task 4: Tasks Done 96
5.2.1 Analysis of the Bolted Beam Splice Module 97
5.2.2 Analysis of the Welded Beam Splice Module 98
5.3 Task 4: Documentation 100

6 Conclusions 101

6.1 Tasks Accomplished 101
6.2 Skills Developed 101

A Appendix 105

A.1 Work Reports 105

Bibliography 112

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.

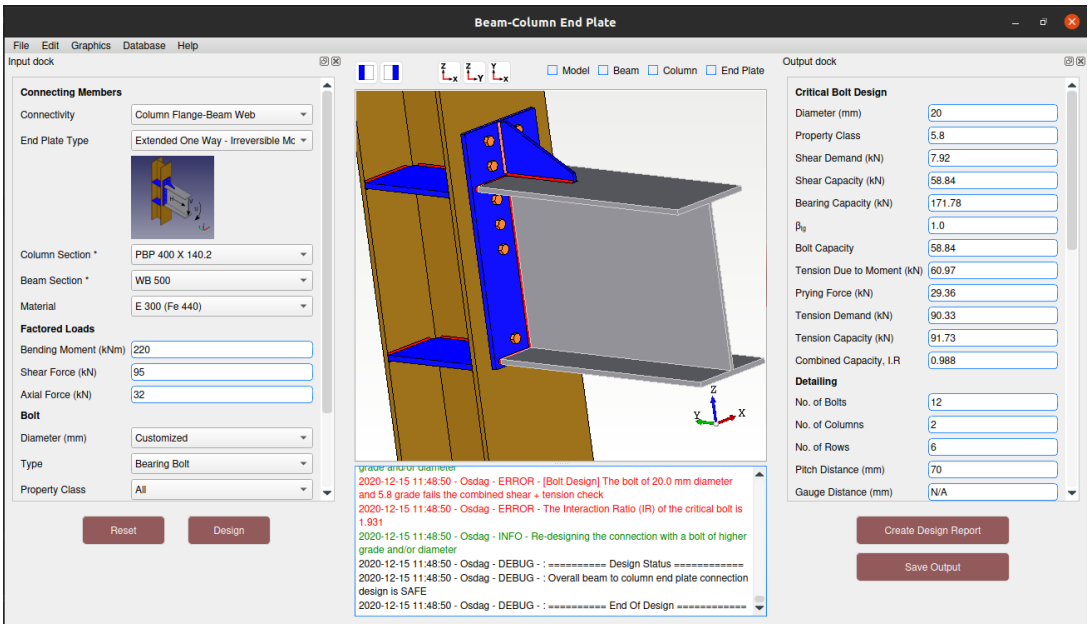


Figure 1.2: Osdag GUI

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.

Screening Task

Problem Statement

Analysis of simply supported beams under moving load using the influence line diagram.
Project Repository: `https://github.com/nishikantmandal007/Beam_ILD_Analysis`

Tasks Done

To address the problem statement, a complete desktop application was developed in Python. The project's execution involved a multi-faceted approach, combining structural engineering principles with modern software development practices to create a robust, user-friendly, and accurate analysis tool. The methodology focused on creating a modular and scalable application, from the core calculation engine to the final user interface.

Core Analysis Using Influence Line Diagrams

The analytical heart of the program was built upon the fundamental structural engineering principle of the Influence Line Diagram (ILD). This method was chosen for its effectiveness in determining the maximum stress effects from moving loads. The core logic involves:

- **Implementation of ILD Ordinates:** Functions were coded to accurately calculate the ILD ordinates for support reactions, shear force, and bending moment at any given point along the simply supported beam.
- **Handling Multiple Load Types:** The engine was designed to be versatile, capable of analyzing two distinct and critical load cases: a system of two moving point

loads at a fixed distance apart, and a moving Uniformly Distributed Load (UDL) of a specified length and intensity.

- **Calculation of Absolute Maximums:** By applying the ILD principle, the program systematically determines the critical positions of the load systems that result in the absolute maximum support reactions, shear force, and bending moment anywhere in the beam, providing the essential values required for a safe design.

Graphical User Interface (GUI) Development

A key objective was to make the analysis tool accessible and intuitive. To achieve this, a sophisticated Graphical User Interface was designed and built from the ground up using the powerful PyQt6 framework.

- **Intuitive Input Panel:** The GUI features a clean input panel where users can easily define all necessary parameters, including beam length, load type, and the specific magnitudes and dimensions of the loads. Input fields are equipped with real-time validators to prevent erroneous data entry (e.g., negative lengths).

- **Dynamic Interface:** The interface intelligently adapts to the user's selections. For instance, choosing the "Two Point Loads" option displays the relevant input fields for that system, while selecting "UDL" hides them and shows the UDL-specific fields, preventing user confusion.

- **Tabbed Output Display:** The results are organized neatly into a tabbed widget, separating the formatted numerical summary from the graphical plots, allowing users to focus on the information they need.

Visualization and Dynamic Plotting

To provide deeper insight beyond numerical results, powerful visualization capabilities were integrated using the Matplotlib library.

- **Dynamic ILD Plots:** The application dynamically generates and displays the Influence Line Diagrams for the reactions at both supports, as well as for the shear force and bending moment at the mid-span.

- **Embedded Graphics:** These plots are not generated in a separate window but are seamlessly embedded within the GUI's "Influence Lines" tab, creating a cohesive and professional user experience. This visual feedback allows engineers and students to immediately understand the influence of a load's position on the internal forces of the beam.

Software Architecture and Design

The program was intentionally structured using Object-Oriented Programming (OOP) principles to ensure the codebase is clean, modular, and scalable.

- **Logical Class Structure:** The code is organized into logical classes such as `Beam`, `PointLoadSystem`, `UDLLoadSystem`, and `AnalysisResults`. This separation of concerns makes the code easier to read, debug, and extend with new features in the future.

- **Error Handling:** A robust error-handling mechanism was implemented. It catches invalid inputs and calculation errors, presenting clear, user-friendly warning messages through the GUI instead of crashing the application, thus ensuring a stable and reliable tool.

Chapter 2

Internship Task 1: Development of Design and Detailing Checklists for Welded Connections

2.1 Task 1: Problem Statement

The primary objective of this task was to develop comprehensive Design and Detailing Checklists (DDCL) for two fundamental types of welded connections within the Osdag software framework: **Welded Lap Joints** and **Welded Butt Joints**. Osdag, which stands for Open Steel Design and Graphics, aims to provide an open-source tool for the design and detailing of steel structures in compliance with Indian Standards. A critical requirement for ensuring the software's accuracy, transparency, and usability is the creation of detailed documentation that outlines the underlying design philosophy and codal provisions. These DDCLs serve as the backbone for the software's design logic, acting as a reference for both developers during implementation and for end-users seeking to understand the design process. The problem was to meticulously research, structure, and document all the necessary steps, formulas, and compliance checks as stipulated by **IS 800:2007** for these two connection types.

2.2 Task 1: Tasks Done

To address the problem statement, a systematic methodology was adopted to create the DDCLs for both welded lap and butt joints. The process involved a deep dive into the relevant engineering codes and structuring the information in a logical, step-by-step format that could be easily translated into software algorithms and understood by civil engineering professionals.

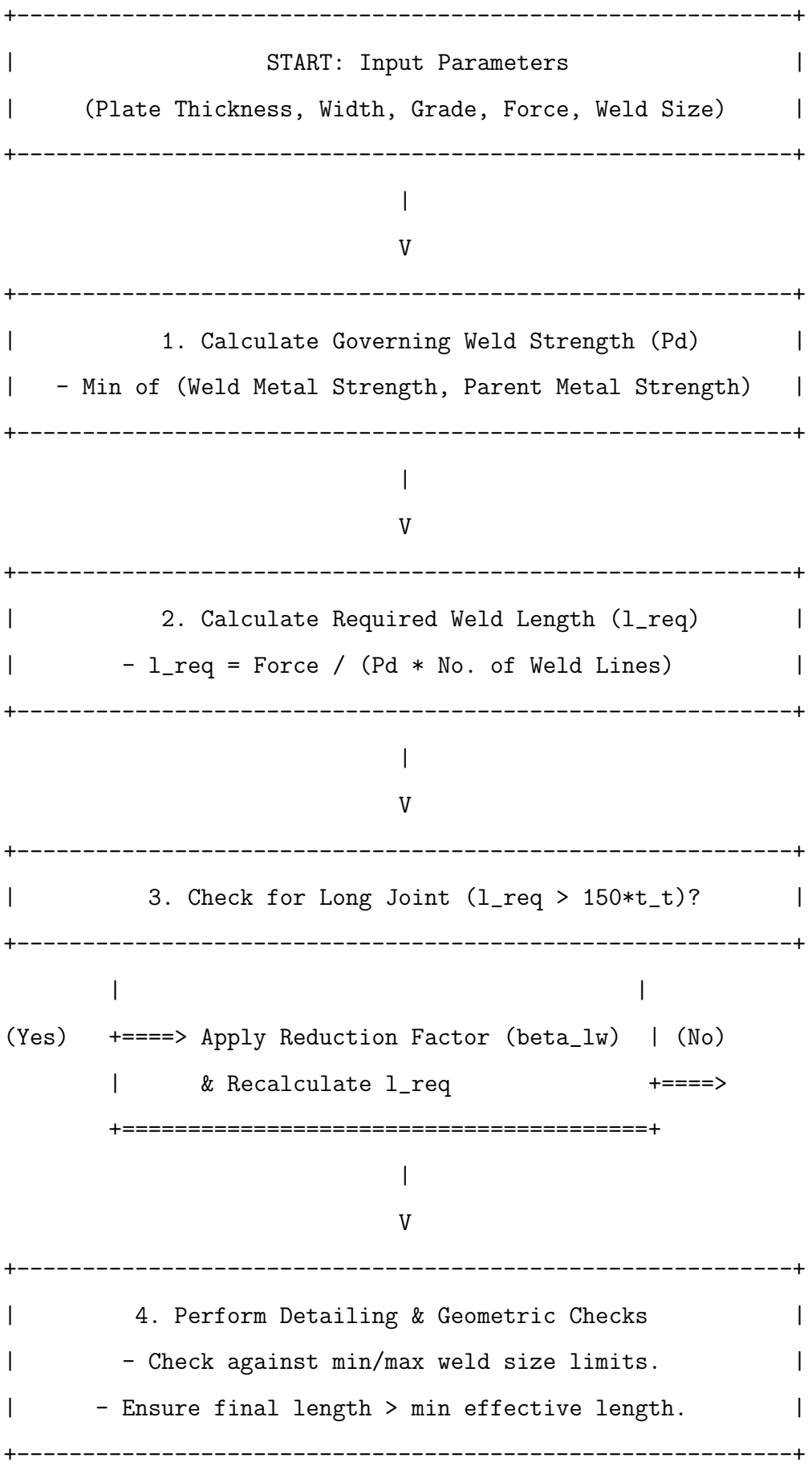
2.2.1 Methodology and Implementation

The core of the task involved a thorough review of the Indian Standard IS 800:2007, specifically focusing on the clauses pertaining to the design of welded connections. The key activities included:

- **Identification of Input and Output Parameters:** The first step was to identify all the essential data that a user must provide to design a connection (e.g., plate thickness, material grade, factored load) and the results the software should generate (e.g., weld size, weld length, utilization ratio). This ensures a clear definition of the user interface and the design output.
- **Formulation of Design Logic:** The complete design process was broken down into logical steps. This involved extracting all relevant formulas and conditions from IS 800:2007 for calculating weld strength, required weld dimensions, and various reduction factors.
- **Structuring the DDCL:** The information was organized into a clear checklist format. This included sections on theory, input/output parameters, detailed design calculations, and detailing requirements, making the documents comprehensive and easy to follow.

2.2.2 Development of DDCL for Welded Lap Joint

For the welded lap joint module, the DDCL was developed to guide the design of overlapping plates connected by fillet welds. My work involved detailing the entire design and verification process, which is summarized in the flowchart below.


START: Input Parameters
(Plate Thickness, Width, Grade, Force, Weld Size)
1. Calculate Governing Weld Strength (Pd)
- Min of (Weld Metal Strength, Parent Metal Strength)
2. Calculate Required Weld Length (l_req)
- l_req = Force / (Pd * No. of Weld Lines)
3. Check for Long Joint (l_req > 150*t_t)?
(Yes)
Apply Reduction Factor (beta_lw)
& Recalculate l_req
(No)
4. Perform Detailing & Geometric Checks
- Check against min/max weld size limits.
- Ensure final length > min effective length.

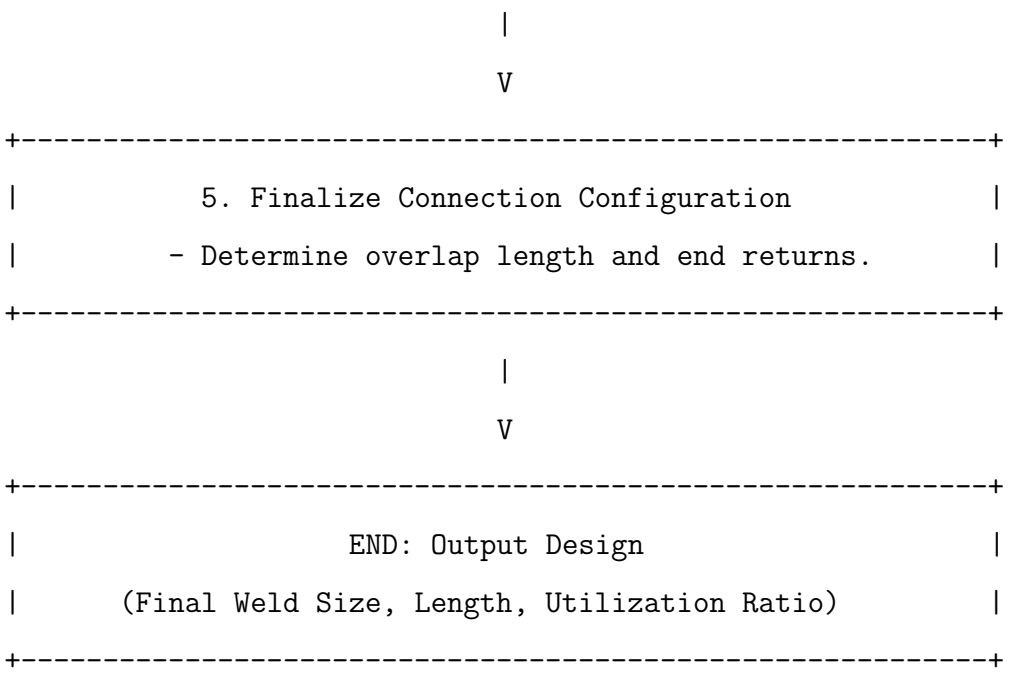
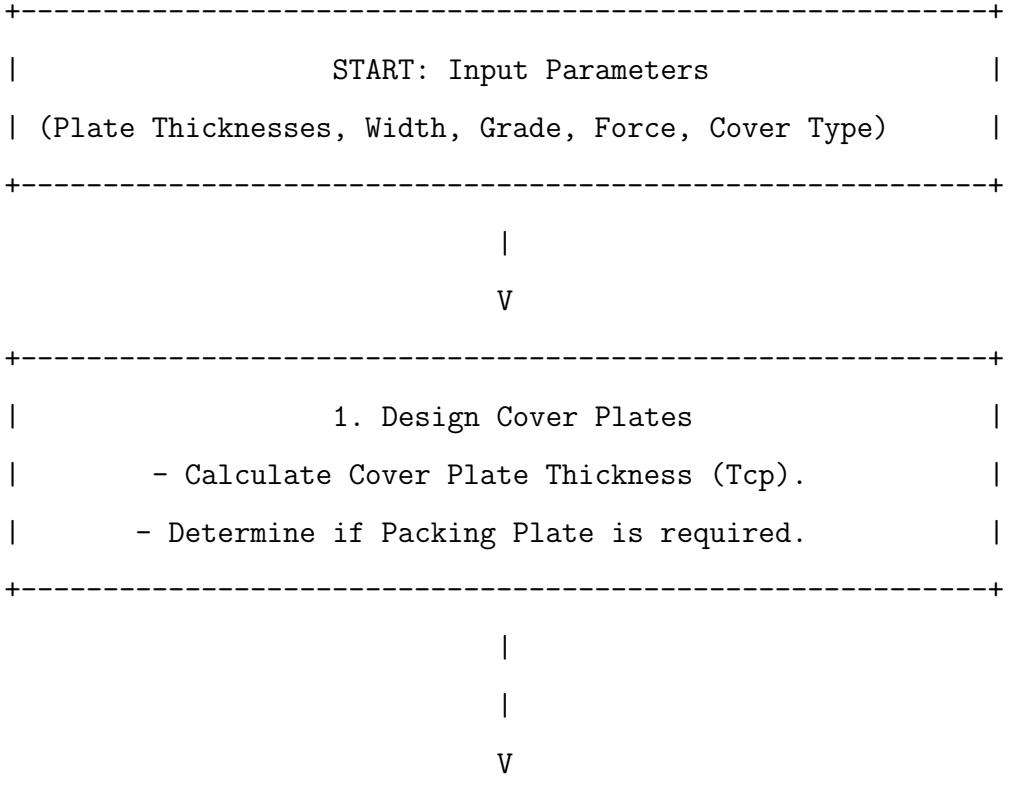


Figure 1: Algorithmic Flow for Welded Lap Joint Design.

2.2.3 Development of DDCL for Welded Butt Joint

For the more complex welded butt joint, the DDCL covers end-to-end connections, often utilizing cover plates and fillet welds. My work involved structuring the design stages as illustrated in the following algorithmic flow.



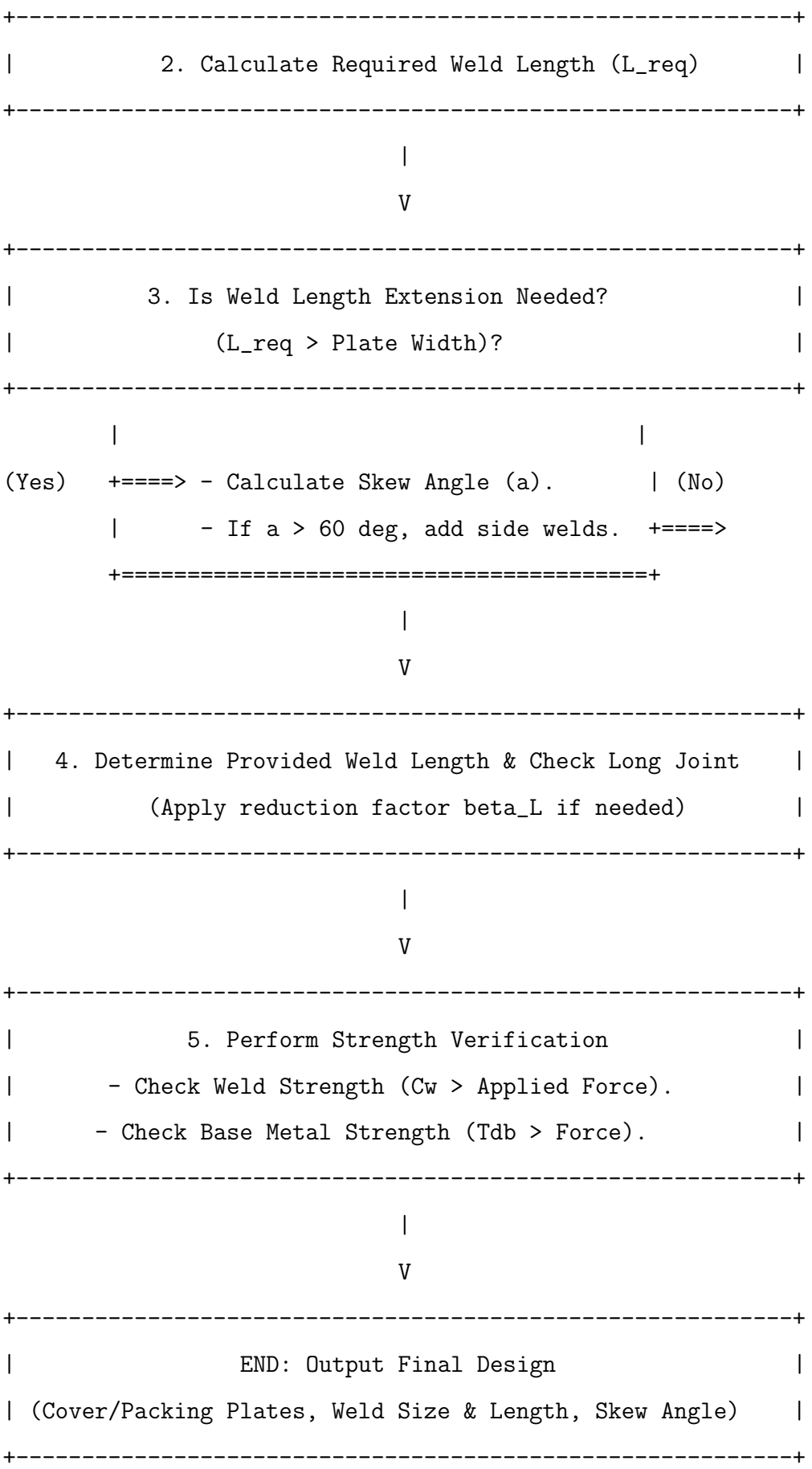

2. Calculate Required Weld Length (L_req)
3. Is Weld Length Extension Needed?
(L_req > Plate Width)?
(Yes)
- Calculate Skew Angle (a).
- If a > 60 deg, add side welds.
(No)
4. Determine Provided Weld Length & Check Long Joint
(Apply reduction factor beta_L if needed)
5. Perform Strength Verification
- Check Weld Strength (Cw > Applied Force).
- Check Base Metal Strength (Tdb > Force).
END: Output Final Design
(Cover/Packing Plates, Weld Size & Length, Skew Angle)

Figure 2: Algorithmic Flow for Welded Butt Joint Design.

2.3 Task 1: Documentation

My contribution towards the Osdag Developer/User manual was the creation of these two DDCL documents. These documents are more than just internal checklists; they form a crucial part of the official documentation. They provide a transparent, step-by-step breakdown of the engineering principles and codal provisions that the software modules are built upon.

2.3.1 Purpose of the DDCL in Osdag

The DDCLs I developed for the Welded Lap and Butt Joints serve several key functions:

- **For Developers:** They act as a precise blueprint for writing and verifying the Python code for the design modules. By following the checklist, developers can ensure that all calculations and logical checks are correctly implemented according to IS 800:2007.
- **For Users:** They provide complete transparency into the design process. An engineer using Osdag can refer to the DDCL to understand exactly how the software arrived at a particular weld size or plate dimension, building trust and confidence in the tool.
- **For Verification and Validation:** These documents are essential for calibrating and testing the software. The verification examples for Osdag can be manually calculated using the steps in the DDCL and compared against the software's output to ensure accuracy.
- **For Education:** The DDCLs serve as excellent educational resources for students and junior engineers to learn the practical application of IS 800:2007 for designing steel connections.

By authoring these comprehensive documents, I have helped ensure that the Welded Lap and Butt Joint modules in Osdag are robust, reliable, and transparent, aligning with the project's open-source philosophy.

Chapter 3

Internship Task 2: Implementation of the Welded Butt Joint Design Module

This module was collaboratively developed by Nishi Kant Mandal (`https://github.com/nishikantmandal007`) and Tanu Singh (`https://github.com/Tanu-Singh-22`) during the fellowship period.

3.1 Task 2: Problem Statement

The primary objective of this task was to develop a new, fully functional design module for **Welded Butt Joints** and integrate it into the Osdag software framework. The goal was to create a tool that automates the complex design process for this connection type in strict accordance with the Indian Standard IS 800:2007. The module needed to be robust enough to handle various user inputs, including different plate dimensions, material grades, and factored tensile loads. A critical requirement was to implement the detailed engineering logic for calculating cover plate dimensions, required weld lengths (including provisions for skewed welds), and verifying the strength of all components to provide a clear "SAFE" or "UNSAFE" design status to the user.

3.2 Task 2: Tasks Done

The development of the Welded Butt Joint module was a comprehensive process that involved translating complex engineering standards into a functional, user-friendly software component. The implementation followed a structured approach, ensuring that the module was not only accurate in its calculations but also seamlessly integrated into the existing Osdag architecture.

3.2.1 Module Architecture and Integration

The foundation of the task was to structure the new module correctly within the Osdag framework.

- A new Python class, `ButtJointWelded`, was created. It inherits from the base `MomentConnection` class, ensuring it adheres to the established design patterns and interfaces used across the Osdag platform.
- The module was equipped with functions to define its user interface, including the main input fields (`input_values`), design preference tabs for weld and detailing options (`weld_values`, `detailing_values`), and the final output display (`output_values`).

3.2.2 Core Design Logic Implementation

The most critical part of the task was the implementation of the complete design-and-check sequence as per IS 800:2007. This logic was built methodically, mirroring the steps outlined in the Design and Detailing Checklist (DDCL).

- **Input Processing:** The `set_input_values` function was developed to capture all user-defined parameters from the GUI, initialize core objects like `Plate` and `Weld`, and calculate initial parameters such as the required cover plate and packing plate thicknesses.
- **Weld Design Sequence:** A main orchestrator function, `design_of_weld`, was implemented to control the entire design flow. This function is responsible for

determining a valid weld size based on user selection and code constraints before proceeding to the detailed checks.

- **Weld Length and Geometry:** The `weld_length` function contains the logic to calculate the required weld length. A key feature implemented here is the automated handling of cases where the required length exceeds the plate width, triggering calculations for skewed welds, including determining the necessary skew angle and any additional side weld lengths required.

- **Strength and Safety Verification:** A series of verification functions were coded to check every aspect of the design's safety. This includes `weld_strength_verification` to check the weld capacity, `long_joint_reduction_factor` to apply reductions for long welds, and `check_base_metal_strength` to ensure the connected plates themselves will not fail under tension.

- **Final Status Determination:** The `calculate_final_utilization_ratio` function aggregates the stress ratios from all component checks, identifies the maximum (or critical) utilization ratio, and sets the final `design_status` to either SAFE or UNSAFE.

3.2.3 Output and Reporting

To communicate the design results back to the user, comprehensive output and reporting functionalities were created.

- The `output_values` function populates the GUI with the calculated design parameters, such as the provided cover plate dimensions, the final weld size and length, and the overall utilization ratio.

- A detailed design report generation feature was implemented via the `save_design` function, which organizes all input parameters, design checks, and final results into a professional LaTeX report for documentation and submission purposes.

3.3 Task 2: Python Code

The entire logic for the Welded Butt Joint module is encapsulated within the `butt_joint_welded.py` file. This script is a class-based implementation that follows Osdag's established software architecture. Below are key snippets from the module with explanations of their functionality.

3.3.1 Description of the Script

The script defines the `ButtJointWelded` class, which handles everything from user input to final design report generation. It uses helper objects like `Plate` and `Weld` to manage properties and perform calculations. The design flow is orchestrated through a series of dedicated methods, each responsible for a specific clause or check from IS 800:2007.

3.3.2 Code Snippets and Explanation

The following snippets showcase the core logic implemented in the module.

Listing 3.1: Initialization and Cover Plate Logic

```
1  def set_input_values(self, design_dictionary):
2      # ... (omitted initial setup for brevity) ...
3      self.plate1 = Plate(thickness=[design_dictionary[
           KEY_PLATE1_THICKNESS]],
4                          material_grade=design_dictionary[KEY_MATERIAL],
5                          width=design_dictionary[KEY_PLATE_WIDTH])
6      self.plate2 = Plate(thickness=[design_dictionary[
           KEY_PLATE2_THICKNESS]],
7                              material_grade=design_dictionary[KEY_MATERIAL],
8                              width=design_dictionary[KEY_PLATE_WIDTH])
9
10     self.weld = Weld(material_g_o=design_dictionary[
           KEY_DP_WELD_MATERIAL_G_O],
11                       type=design_dictionary[KEY_DP_WELD_TYPE],
12                       fabrication=design_dictionary.get(KEY_DP_FAB_SHOP,
                             KEY_DP_FAB_SHOP))
13
14     plate1_thk = float(design_dictionary[KEY_PLATE1_THICKNESS])
```

```
    plate2_thk = float(design_dictionary[KEY_PLATE2_THICKNESS])
    Tmin = min(plate1_thk, plate2_thk)
    cover_plate_type_str = design_dictionary[KEY_COVER_PLATE]

    available_thicknesses = [float(thk) for thk in PLATE_THICKNESS_SAIL
        ]
    if "double" in cover_plate_type_str.lower():
        self.planes = 2
        Tcp = math.ceil((9.0 / 16.0) * Tmin) # Double cover
        self.calculated_cover_plate_thickness = min(
            [thk for thk in available_thicknesses if thk >= Tcp],
                default=Tcp)
        # Packing plate logic
        if abs(plate1_thk - plate2_thk) > 0.001:
            self.packing_plate_thickness = abs(plate1_thk - plate2_thk)
        else:
            self.packing_plate_thickness = 0.0
    elif "single" in cover_plate_type_str.lower():
        self.planes = 1
        Tcp = math.ceil((5.0 / 8.0) * Tmin) # Single cover
        self.calculated_cover_plate_thickness = min(
            [thk for thk in available_thicknesses if thk >= Tcp],
                default=Tcp)
        self.packing_plate_thickness = 0.0
    # ...
    self.design_of_weld(self,design_dictionary)
```

Explanation: This function is the entry point for the design process. It takes the user's input dictionary, initializes the `Plate` and `Weld` objects, and then immediately performs the crucial initial calculation for the cover plate thickness based on whether a single or double cover is selected. It also implements the logic for determining if a packing plate is needed when the two main plates have different thicknesses.

Listing 3.2: Weld Length and Skew Angle Calculation

```
def weld_length(self, design_dictionary):
    # ... (omitted initial setup) ...
    if "single" in self.cover_plate.lower():
        self.N_f = 1  # Number of welds
    else:
```

```
6           self.N_f = 2  # Double cover plate
7
8       # Calculate required weld length
9       self.L_req = self.tensile_force / (self.N_f * 0.707 * self.
            weld_size * self.f_w)
10
11      # Check if straight weld is sufficient
12      if self.L_req <= self.plates_width:
13          self.weld_length_provided = self.plates_width
14          self.weld_angle = 0
15      else:
16          # Calculate skewed weld parameters
17          L_target = self.L_req / self.N_f
18
19          # Calculate skew angle
20          self.weld_angle = math.degrees(math.atan((L_target - self.
                plates_width)/(2 * self.plates_width)))
21
22          # Constrain angle between 20-60 degrees
23          if self.weld_angle < 20: self.weld_angle = 20
24          elif self.weld_angle > 60: self.weld_angle = 60
25
26          # Calculate provided length per weld line with skew
27          L_provided_line = self.plates_width + 2 * self.plates_width *
                math.tan(math.radians(self.weld_angle))
28          L_provided_total = self.N_f * L_provided_line
29
30          # Check if side welds are needed
31          if L_provided_total < self.L_req:
32              self.side_weld_length = (self.L_req - L_provided_total) /
                    self.N_f
33          else:
34              self.side_weld_length = 0
```

Explanation: This snippet showcases one of the most complex logical parts of the module. It first calculates the total required length of the fillet weld. If this length is greater than the width of the plate, it automatically calculates the required skew angle of the cover plate to accommodate the extra length. The angle is constrained between

20 and 60 degrees as per good practice, and if even the maximum skew is insufficient, it calculates the additional length that must be provided via side welds.

Listing 3.3: Final Design Status Determination

```
def calculate_final_utilization_ratio(self):
    """Calculate final utilization ratio and set design status after
       all component checks"""
    if not hasattr(self, 'utilization_ratios'):
        self.design_status = False
        return

    # Get maximum utilization ratio across all components
    self.utilization_ratio = max(self.utilization_ratios.values())

    # Design is safe only if all utilization ratios are < 1.0
    if self.utilization_ratio > 1.0:
        self.design_status = False
        critical_component = max(self.utilization_ratios.items(), key=
           lambda x: x[1])[0]
        # ... (logging errors and recommendations) ...
    else:
        self.design_status = True
```

Explanation: After all individual checks (weld strength, base metal strength, etc.) are performed, this function determines the overall safety of the connection. It finds the highest (most critical) utilization ratio among all checked components. If this value is greater than 1.0, the design is marked as "UNSAFE"; otherwise, it is marked as "SAFE". This provides a single, unambiguous result to the user.

3.3.3 Full code

The following command imports the entire source code for the module directly into the report for complete reference. This assumes the file `butt_joint_welded.py` is placed in a `codes` sub-directory.

```
"""
Module: butt_joint_welded.py
Author: Aman, Tanu Singh, Nishi Kant Mandal
```

```
Date: 2025-06-12

Description:
    ButtJointWelded is a moment connection module that represents
        a welded butt joint connection.
    It inherits from MomentConnection and follows the same
       structure and design logic as other
    connection modules (e.g., BeamCoverPlate, ColumnCoverPlate)
       used in Osdag.

Reference:
    - Osdag software guidelines and connection module structure
       documentation
"""

from .moment_connection import MomentConnection
from ...utils.common.component import *
from ...utils.common.is800_2007 import IS800_2007
from ...utils.common.is800_2007 import *
from ...Common import *
from ...design_report.reportGenerator_latex import CreateLatex
from ...Report_functions import *
from ...utils.common.load import Load
import logging

import math

from PyQt5.QtCore import Qt

class ButtJointWelded(MomentConnection):
    def __init__(self):
        super(ButtJointWelded, self).__init__()
        self.design_status = False
        self.weld_size = None
```

```
        self.weld_length_provided = None
        self.weld_strength = None
        self.weld_thickness = None
        self.plate_width = None
        self.plate_length = None
        self.plate_thickness = None
        self.weld_type = None
        self.weld_material = None
        self.weld_fabrication = None
        self.weld_angle = None
        self.weld_length_effective = None


    ################################################
    # Design Preference Functions Start
    ################################################
    def tab_list(self):
        tabs = []
        tabs.append((("Weld", TYPE_TAB_2, self.weld_values))) #
            added this line t.s.
        tabs.append(("Detailing", TYPE_TAB_2, self.
            detailing_values))
        return tabs

    def tab_value_changed(self):
        # No tab value dependencies needed for bolt and detailing
        return []

    def edit_tabs(self):
        return []  # Keep original empty implementation

    def input_dictionary_design_pref(self):
        design_input = []
        design_input.append(("Weld", TYPE_COMBOBOX, [
```

```
            KEY_DP_WELD_TYPE,
            KEY_DP_WELD_MATERIAL_G_O
        ]))
        design_input.append(("Detailing", TYPE_COMBOBOX, [
            KEY_DP_DETAILING_EDGE_TYPE,
            KEY_DP_DETAILING_PACKING_PLATE
        ]))
        return design_input

    def input_dictionary_without_design_pref(self):
        design_input = []
        design_input.append((None, [
            KEY_DP_WELD_TYPE,
            KEY_DP_WELD_MATERIAL_G_O,
            KEY_DP_DETAILING_EDGE_TYPE,
            KEY_DP_DETAILING_PACKING_PLATE
        ], ''))
        return design_input

    def get_values_for_design_pref(self, key, design_dictionary):
        # Get fu value from selected material
        if design_dictionary[KEY_MATERIAL] != 'Select Material':
            fu = Material(design_dictionary[KEY_MATERIAL], 41).fu
        else:
            fu = ''

        # Default values as per requirements
        defaults = {
            KEY_DP_WELD_TYPE: "Shop weld",
            KEY_DP_WELD_MATERIAL_G_O: str(fu),  # Set weld
               material grade to fu of selected material
            KEY_DP_DETAILING_EDGE_TYPE: "Sheared or hand flame
               cut",
            KEY_DP_DETAILING_PACKING_PLATE: "Yes"
```

```
        }
        return defaults.get(key)

    def detailing_values(self, input_dictionary):
        values = {
            KEY_DP_DETAILING_EDGE_TYPE: 'Sheared or hand flame
               cut',
            KEY_DP_DETAILING_PACKING_PLATE: 'Yes',
        }

        for key in values.keys():
            if key in input_dictionary.keys():
                values[key] = input_dictionary[key]

        detailing = []

        # Edge preparation method as per Cl. 10.2.4 of IS
           :800:2007
        t1 = (KEY_DP_DETAILING_EDGE_TYPE,
           KEY_DISP_DP_DETAILING_EDGE_TYPE, TYPE_COMBOBOX,
            ['Sheared or hand flame cut', 'Rolled, machine-flame
               cut, sawn and planed'],
            values[KEY_DP_DETAILING_EDGE_TYPE])
        detailing.append(t1)

        t49 = (KEY_DP_DETAILING_PACKING_PLATE,
           KEY_DISP_DP_DETAILING_PACKING_PLATE, TYPE_COMBOBOX,
               ['Yes', 'No'], values[
                  KEY_DP_DETAILING_PACKING_PLATE])
        detailing.append(t49)

        t4 = ("textBrowser", "", TYPE_TEXT_BROWSER,
           DETAILING_DESCRIPTION_LAPJOINT, None)
        detailing.append(t4)
```

```
        return detailing

    # def bolt_values(self, input_dictionary):
    #     values = {
    #         KEY_DP_BOLT_TYPE: 'Non Pre-tensioned',
    #         KEY_DP_BOLT_HOLE_TYPE: 'Standard',
    #         KEY_DP_BOLT_SLIP_FACTOR: '0.3'
    #     }

    #     for key in values.keys():
    #         if key in input_dictionary.keys():
    #             values[key] = input_dictionary[key]

    #     bolt = []

    #     # Bolt type selection
    #     t1 = (KEY_DP_BOLT_TYPE, "Type", TYPE_COMBOBOX,
    #         ['Non Pre-tensioned', 'Pre-tensioned'],
    #         values[KEY_DP_BOLT_TYPE])
    #     bolt.append(t1)

    #     # Bolt hole type
    #     t2 = (KEY_DP_BOLT_HOLE_TYPE, "Bolt Hole", TYPE_COMBOBOX
    ,
    #         ['Standard', 'Over-sized'],
    #         values[KEY_DP_BOLT_HOLE_TYPE])
    #     bolt.append(t2)

    #     # Slip factor as per Table 20 of IS 800
    #     t3 = (KEY_DP_BOLT_SLIP_FACTOR, "Slip Factor",
    TYPE_COMBOBOX,
    #         ['0.3', '0.45', '0.5'],
    #         values[KEY_DP_BOLT_SLIP_FACTOR])
```

```
    #     bolt.append(t3)

    #     return bolt

    # added weld function

    def weld_values(self, input_dictionary):
        # Get fu value from selected material if available
        fu = ''
        if input_dictionary and KEY_MATERIAL in input_dictionary:
            if input_dictionary[KEY_MATERIAL] != 'Select Material
               ':
                fu = Material(input_dictionary[KEY_MATERIAL], 41)
                   .fu

        values = {
            KEY_DP_WELD_TYPE: 'Shop weld',
            KEY_DP_WELD_MATERIAL_G_O: str(fu) if fu else '410',
               # Default to 410 if no material selected
        }

        # Update values from input dictionary if available
        for key in values.keys():
            if input_dictionary and key in input_dictionary:
                values[key] = input_dictionary[key]

        weld = []

        t3 = (KEY_DP_WELD_TYPE, "Type", TYPE_COMBOBOX,
            ['Shop weld', 'Field weld'],
            values[KEY_DP_WELD_TYPE])
        weld.append(t3)

        t2 = (KEY_DP_WELD_MATERIAL_G_O, "Material Grade Overwrite
```

```
            , Fu (MPa)", TYPE_TEXTBOX,
             None,
             values[KEY_DP_WELD_MATERIAL_G_O])
        weld.append(t2)
        return weld


    ###################################
    # Design Preference Functions End
    ###################################

    def set_osdaglogger(key):

        """
        Function to set Logger for Tension Module
        """

        # @author Arsil Zunzunia
        global logger
        logger = logging.getLogger('Osdag')

        logger.setLevel(logging.DEBUG)
        handler = logging.StreamHandler()
        formatter = logging.Formatter(fmt='%(asctime)s - %(name)s
            - %(levelname)s - %(message)s', datefmt='%Y-%m-%d %H
           :%M:%S')

        handler.setFormatter(formatter)
        logger.addHandler(handler)
        handler = logging.FileHandler('logging_text.log')

        formatter = logging.Formatter(fmt='%(asctime)s - %(name)s
            - %(levelname)s - %(message)s', datefmt='%Y-%m-%d %H
           :%M:%S')
```

```
        handler.setFormatter(formatter)
        logger.addHandler(handler)

        if key is not None:
            handler = OurLog(key)
            formatter = logging.Formatter(fmt='%(asctime)s - %(
               name)s - %(levelname)s - %(message)s',
                                          datefmt='%Y-%m-%d %H:%M
                                             :%S')
            handler.setFormatter(formatter)
            logger.addHandler(handler)


    def input_value_changed(self):

        lst = []

        t8 = ([KEY_MATERIAL], KEY_MATERIAL, TYPE_CUSTOM_MATERIAL,
            self.new_material)
        lst.append(t8)

        return lst
    # not sure about this function no changes done here -t.s.

    def input_values(self):
        options_list = []

        t16 = (KEY_MODULE, KEY_DISP_BUTTJOINTWELDED, TYPE_MODULE,
            None, True, 'No Validator')
        options_list.append(t16)

        t5 = (None, DISP_TITLE_CM, TYPE_TITLE, None, True, 'No
           Validator')
        options_list.append(t5)
```

```
        t31 = (KEY_PLATE1_THICKNESS, KEY_DISP_PLATE1_THICKNESS,
            TYPE_COMBOBOX, VALUES_PLATETHK_CUSTOMIZED, True, 'Int
            Validator')
        options_list.append(t31)

        t34 = (KEY_PLATE2_THICKNESS, KEY_DISP_PLATE2_THICKNESS,
            TYPE_COMBOBOX, VALUES_PLATETHK_CUSTOMIZED, True, 'Int
            Validator')
        options_list.append(t34)

        t35 = (KEY_PLATE_WIDTH, KEY_DISP_PLATE_WIDTH,
            TYPE_TEXTBOX, None, True, 'Float Validator')
        options_list.append(t35)

        t6 = (KEY_MATERIAL, KEY_DISP_MATERIAL, TYPE_COMBOBOX,
            VALUES_MATERIAL, True, 'No Validator')
        options_list.append(t6)

        t36 = (KEY_COVER_PLATE, KEY_DISP_COVER_PLT, TYPE_COMBOBOX
            , VALUES_COVER_PLATE, True, 'No Validator')
        options_list.append(t36)

        t18 = (None, DISP_TITLE_WELD, TYPE_TITLE, None, True, 'No
             Validator')
        options_list.append(t18)

        t20 = (KEY_WELD_SIZE, KEY_DISP_WELD_SIZE,
            TYPE_COMBOBOX_CUSTOMIZED, VALUES_ALL_CUSTOMIZED, True,
             'No Validator')
        options_list.append(t20)

        t7 = (None, DISP_TITLE_FSL, TYPE_TITLE, None, True, 'No
            Validator')
```

```
        options_list.append(t7)

        t17 = (KEY_TENSILE_FORCE, KEY_DISP_TENSILE_FORCE,
           TYPE_TEXTBOX, None, True, 'Int Validator')
        options_list.append(t17)

        return options_list

    def customized_input(self):
        list1 = []
        t11 = (KEY_WELD_SIZE, self.weld_size_customized)
        list1.append(t11)
        return list1

    @staticmethod
    def weld_size_customized():
        return [str(size) for size in WELD_SIZES]

    def spacing(self, status):
        spacing = []

        t00 = (None, "", TYPE_NOTE, "Representative Image for
           Spacing Details - 3 x 3 pattern considered")
        spacing.append(t00)

        t99 = (None, 'Spacing Details', TYPE_SECTION,
            [str(files("osdag.data.ResourceFiles.images").
               joinpath("spacing_3.png")), 400, 277, ""])  # [
               image, width, height, caption]
        spacing.append(t99)

        t9 = (KEY_OUT_PITCH, KEY_OUT_DISP_PITCH, TYPE_TEXTBOX,
           self.plate.gauge_provided if status else '')
        spacing.append(t9)
```

```
        t10 = (KEY_OUT_END_DIST, KEY_OUT_DISP_END_DIST,
            TYPE_TEXTBOX, self.plate.edge_dist_provided if status
            else '')
        spacing.append(t10)

        t111 = (KEY_OUT_GAUGE, KEY_OUT_DISP_GAUGE, TYPE_TEXTBOX,
            self.plate.pitch_provided if status else '')
        spacing.append(t111)

        t12 = (KEY_OUT_EDGE_DIST, KEY_OUT_DISP_EDGE_DIST,
            TYPE_TEXTBOX, self.plate.end_dist_provided if status
            else '')
        spacing.append(t12)

        return spacing

    def output_values(self, flag):
        out_list=[]
        # Cover plate details
        t44 = (None, DISP_TITLE_COVER_PLATE, TYPE_TITLE, None,
            True)
        out_list.append(t44)

        t22 = (KEY_OUT_UTILISATION_RATIO,
            KEY_OUT_DISP_UTILISATION_RATIO, TYPE_TEXTBOX,
                round(self.utilization_ratio, 3) if flag else '',
                    True)
        out_list.append(t22)

        # Calculate cover_type based on planes attribute
        cover_type = ''
        if flag and hasattr(self, 'planes'):
            cover_type = "Double" if self.planes == 2 else "
```

```
            Single"

        t13 = (KEY_OUT_NO_COVER_PLATE,
           KEY_OUT_DISP_NO_COVER_PLATE, TYPE_TEXTBOX,
               cover_type if flag else '', True)
        out_list.append(t13)

        t38 = (KEY_OUT_WIDTH_COVER_PLATE,
           KEY_OUT_DISP_WIDTH_COVER_PLATE, TYPE_TEXTBOX,
               self.plates_width if flag else '', True)
        out_list.append(t38)

        t28 = (KEY_OUT_LENGTH_COVER_PLATE,
           KEY_OUT_DISP_LENGTH_COVER_PLATE, TYPE_TEXTBOX,
               round(self.weld_length_provided, 1) if flag else '
                  ', True)
        out_list.append(t28)

        t47 = (KEY_OUT_THICKNESS_COVER_PLATE,
           KEY_OUT_DISP_THICKNESS_COVER_PLATE, TYPE_TEXTBOX,
               self.calculated_cover_plate_thickness if flag else
                   '', True)
        out_list.append(t47)

        if hasattr(self, 'packing_thickness') and self.
           packing_thickness > 0:
            t15 = (KEY_PK_PLTHK, KEY_DISP_PK_PLTHK, TYPE_TEXTBOX,
                  round(self.packing_thickness, 1) if flag else '
                     ', True)
            out_list.append(t15)

        # Weld details
        t21 = (None, DISP_TITLE_WELD, TYPE_TITLE, None, True)
        out_list.append(t21)
```

```
        t23 = (KEY_OUT_WELD_TYPE, KEY_OUT_DISP_WELD_TYPE,
           TYPE_TEXTBOX,
               "Fillet" if flag else '', True)
        out_list.append(t23)

        t24 = (KEY_OUT_WELD_SIZE, KEY_OUT_DISP_WELD_SIZE,
           TYPE_TEXTBOX,
               round(self.weld_size, 1) if flag else '', True)
        out_list.append(t24)

        t25 = (KEY_OUT_WELD_STRENGTH,
           KEY_OUT_DISP_WELD_STRENGTH_kN, TYPE_TEXTBOX,
               round(self.weld_strength/1000, 2) if flag else '',
                   True)  # Convert to kN
        out_list.append(t25)

        t26 = (KEY_OUT_WELD_LENGTH_EFF,
           KEY_OUT_DISP_WELD_LENGTH_EFF, TYPE_TEXTBOX,
               round(self.weld_length_effective, 1) if flag else
                  '', True)
        out_list.append(t26)

        t27 = (KEY_OUT_BOLT_CONN_LEN, KEY_OUT_DISP_BOLT_CONN_LEN,
            TYPE_TEXTBOX,
               round(self.weld_length_provided, 1) if flag else '
                  ', True)
        out_list.append(t27)

        return out_list

    def module_name(self):

        return KEY_DISP_BUTTJOINTWELDED
```

```
def call_3DColumn(self, ui, bgcolor):
    # status = self.resultObj['Bolt']['status']
    # if status is True:
    #     self.ui.chkBx_beamSec1.setChecked(Qt.Checked)
    if ui.chkBxCol.isChecked():
        ui.btn3D.setChecked(Qt.Unchecked)
        ui.chkBxCol.setChecked(Qt.Unchecked)
        ui.mytabWidget.setCurrentIndex(0)
    # self.display_3DModel("Beam", bgcolor)
    ui.commLogicObj.display_3DModel("Column", bgcolor)

def get_3d_components(self):
    components = []

    t30 = ('Model', self.call_3DModel)
    components.append(t30)

    t32 = ('Plate1', self.call_3DColumn)
    components.append(t32)

    t37 = ('Plate2', self.call_3DPlate)
    components.append(t37)

    return components

def call_3DPlate(self, ui, bgcolor):
    from PyQt5.QtWidgets import QCheckBox
    from PyQt5.QtCore import Qt
    for chkbox in ui.frame.children():
        if chkbox.objectName() == 'Cover Plate':
            continue
        if isinstance(chkbox, QCheckBox):
            chkbox.setChecked(Qt.Unchecked)
```

```
        ui.commLogicObj.display_3DModel("Cover Plate", bgcolor)


    def func_for_validation(self, design_dictionary):

        all_errors = []
        "check valid inputs and empty inputs in input dock"
        self.design_status = False
        flag = False
        flag1 = False
        flag2 = False

        option_list = self.input_values(self)
        missing_fields_list = []

        # print(f'\n func_for_validation option list = {
            option_list}'
        #       f'\n  design_dictionary {design_dictionary}')

        for option in option_list:
            if option[2] == TYPE_TEXTBOX:
                if design_dictionary[option[0]] == '':

                    missing_fields_list.append(option[1])
                else:
                    if option[2] == TYPE_TEXTBOX and option[0] ==
                        KEY_PLATE_WIDTH:

                        if float(design_dictionary[option[0]]) <=
                            0.0:
                            error = "Input value(s) cannot be
                                equal or less than zero."
                            all_errors.append(error)
                        else:
```

```
                            flag1 = True

                    if option[2] == TYPE_TEXTBOX and option[0] ==
                        KEY_TENSILE_FORCE:

                        if float(design_dictionary[option[0]]) <=
                            0.0:
                            error = "Input value(s) cannot be
                               equal or less than zero."
                            all_errors.append(error)
                        else:
                            flag2 = True
            else:
                pass


        if len(missing_fields_list) > 0:
            error = self.generate_missing_fields_error_string(
               self, missing_fields_list)
            all_errors.append(error)
        else:
            flag = True

        print(f'flag = {flag}, flag1 = {flag1}, flag2 = {flag2}')
        if flag  and flag1 and flag2:
            self.set_input_values(self, design_dictionary)
        else:
            return all_errors

    def set_input_values(self, design_dictionary):
        "initialisation of components required to design a butt
           joint welded along with connection"
        # Call parent class's set_input_values with default
           values if not provided
```

```
        design_dictionary_with_defaults = design_dictionary.copy
            ()
        if KEY_SHEAR not in design_dictionary_with_defaults:
            design_dictionary_with_defaults[KEY_SHEAR] = 0.0  #
                Default shear value if not provided
        if KEY_AXIAL not in design_dictionary_with_defaults:
            design_dictionary_with_defaults[KEY_AXIAL] = 0.0  #
                Default axial value if not provided
        if KEY_MOMENT not in design_dictionary_with_defaults:
            design_dictionary_with_defaults[KEY_MOMENT] = 0.0  #
                Default moment value if not provided

        # Call parent class method correctly
        super(ButtJointWelded, self).set_input_values(self,
            design_dictionary_with_defaults)
        print(design_dictionary,"input values are set. Doing
            preliminary member checks")
        self.module = design_dictionary[KEY_MODULE]
        self.mainmodule = "Butt Joint Welded Connection"

        # self.plate_thickness =
            [3,4,6,8,10,12,14,16,20,22,24,25,26,28,30,32,36,40,45,50,56,63,80

        self.main_material = design_dictionary[KEY_MATERIAL]
        self.tensile_force = float(design_dictionary[
            KEY_TENSILE_FORCE])*1000
        self.width = design_dictionary[KEY_PLATE_WIDTH]

        # print(self.sizelist)
        self.efficiency = 0.0
        self.K = 1
        self.count = 0
        self.plate1 = Plate(thickness=[design_dictionary[
            KEY_PLATE1_THICKNESS]],
```

```
                    material_grade=design_dictionary[
                        KEY_MATERIAL],
                    width=design_dictionary[KEY_PLATE_WIDTH])
        self.plate2 = Plate(thickness=[design_dictionary[
            KEY_PLATE2_THICKNESS]],
                            material_grade=design_dictionary[
                                KEY_MATERIAL],
                            width=design_dictionary[
                                KEY_PLATE_WIDTH])

        self.weld = Weld(material_g_o=design_dictionary[
            KEY_DP_WELD_MATERIAL_G_O],
                         type=design_dictionary[KEY_DP_WELD_TYPE
                             ],
                         fabrication=design_dictionary.get(
                             KEY_DP_FAB_SHOP, KEY_DP_FAB_SHOP))
        # Set weld size after creating the weld object
        self.weld.size = design_dictionary[KEY_WELD_SIZE]
        # Start design process
        print("input values are set. Doing preliminary member
            checks")
        self.member_design_status = False
        self.max_limit_status_1 = False
        self.max_limit_status_2 = False
        self.weld_design_status = False
        self.thick_design_status = False
        self.plate_design_status = False

        plate1_thk = float(design_dictionary[KEY_PLATE1_THICKNESS
            ])
        plate2_thk = float(design_dictionary[KEY_PLATE2_THICKNESS
            ])
        Tmin = min(plate1_thk, plate2_thk)
        cover_plate_type_str = design_dictionary[KEY_COVER_PLATE]
```

```
        # Cover plate and packing plate logic as per
            documentation
        available_thicknesses = [float(thk) for thk in
            PLATE_THICKNESS_SAIL]
        if "double" in cover_plate_type_str.lower():
            self.planes = 2
            Tcp = math.ceil((9.0 / 16.0) * Tmin)  # Double cover
                plate thickness as per Eq. 3.2
            self.calculated_cover_plate_thickness = min(
                [thk for thk in available_thicknesses if thk >=
                    Tcp],
                default=Tcp
            )

            # Packing plate logic as per Cl. 10.3.3.2
            if abs(plate1_thk - plate2_thk) > 0.001:
                self.packing_plate_thickness = abs(plate1_thk -
                    plate2_thk)
            else:
                self.packing_plate_thickness = 0.0

        elif "single" in cover_plate_type_str.lower():
            self.planes = 1
            Tcp = math.ceil((5.0 / 8.0) * Tmin)  # Single cover
                plate thickness as per Eq. 3.1
            self.calculated_cover_plate_thickness = min(
                [thk for thk in available_thicknesses if thk >=
                    Tcp],
                default=Tcp
            )
            self.packing_plate_thickness = 0.0

        else:
```

```
            self.planes = 1
            Tcp = Tmin
            self.calculated_cover_plate_thickness = min(
                [thk for thk in available_thicknesses if thk >=
                   Tcp],
                default=Tcp
            )
            self.packing_plate_thickness = 0.0

        self.leg_size = 0
        self.yield_strength = 0
        self.partial_safety_factor = 0
        self.max_weld_size = 0
        #change from here
        self.final_pitch = 0
        self.final_end_dist = 0
        self.final_edge_dist = 0
        self.final_gauge = 0
        self.rows = 0
        self.cols = 0
        self.len_conn = 0
        self.max_gauge_round = 0
        self.max_pitch_round = 0
        self.utilization_ratio = 0
        self.bij = 0
        self.blg = 0
        self.cover_plate = design_dictionary[KEY_COVER_PLATE]

        # Start design process
        self.design_of_weld(self,design_dictionary)

    #=========================DESIGN OF WELD
        ================================================================
```

```
    def design_of_weld(self, design_dictionary):
        """Design sequence for welded butt joint"""
        logger.info(": ===========  Design for Welded Butt Joint
            ===========")
        logger.info(": Design Approach - IS 800:2007 Clause 10")

        # Track individual utilization ratios
        self.utilization_ratios = {}

        # Get the raw weld size input
        weld_size = design_dictionary[KEY_WELD_SIZE]

        plate1_thk = float(design_dictionary[KEY_PLATE1_THICKNESS
            ])
        plate2_thk = float(design_dictionary[KEY_PLATE2_THICKNESS
            ])
        Tmin = min(plate1_thk, plate2_thk)
        s_min = IS800_2007.cl_10_5_2_3_min_weld_size(plate1_thk,
            plate2_thk)
        s_max = Tmin - 1.5

        # If 'all' is selected, pick the first valid weld size as
            per IS 800:2007
        if isinstance(weld_size, str) and weld_size.lower() == '
            all':
            valid_sizes = [s for s in ALL_WELD_SIZES if s_min <=
                s <= s_max]
            if valid_sizes:
                self.weld_size = float(valid_sizes[0])
            else:
                self.weld_size = None
        else:
            # Customized selection: could be a list of
                My_ListWidgetItem or direct value
```

```
            values_to_process = weld_size
            float_weld_sizes = []
            if isinstance(values_to_process, list):
                for item in values_to_process:
                    try:
                        # Handle My_ListWidgetItem or similar
                           objects
                        if hasattr(item, 'text') and callable(
                           item.text):
                            text_val = item.text()
                            float_weld_sizes.append(float(
                               text_val))
                        elif isinstance(item, (str, int, float)):
                            float_weld_sizes.append(float(item))
                    except Exception:
                        continue
                # Use the first valid customized value within IS
                   limits
                valid_custom = [s for s in float_weld_sizes if
                   s_min <= s <= s_max]
                if valid_custom:
                    self.weld_size = float(valid_custom[0])
                else:
                    self.weld_size = None
            else:
                # Single value (customized)
                try:
                    if hasattr(values_to_process, 'text') and
                       callable(values_to_process.text):
                        self.weld_size = float(values_to_process.
                           text())
                    else:
                        self.weld_size = float(values_to_process)
                    # Check IS limits
```

```
                if not (s_min <= self.weld_size <= s_max):
                    self.weld_size = None
            except Exception:
                self.weld_size = None

    # Ensure weld_size is set before using it
    if self.weld_size is None:
        logger.error(": weld_size is not set. Cannot proceed
           with weld design.")
        self.design_status = False
        logger.error(": Design status: UNSAFE due to missing
           or invalid weld size.")
        logger.info(": ==========End Of Design==========")
        return

    # Return the selected weld size for output if needed
    self.selected_weld_size = self.weld_size

    self.effective_throat_thickness = 0.707 * self.weld_size
    self.fu = float(design_dictionary[
       KEY_DP_WELD_MATERIAL_G_O])
    weld_type = design_dictionary[KEY_DP_WELD_TYPE]
    if weld_type == "Shop weld":
        self.gamma_mw = 1.25
    else:
        self.gamma_mw = 1.50
    self.weld_design_strength = self.fu / (math.sqrt(3) *
       self.gamma_mw)
    self.weld_length(self,design_dictionary)
    self.weld_strength_verification(self,design_dictionary)
    self.long_joint_reduction_factor(self)
    self.check_base_metal_strength(self,design_dictionary)
    self.calculate_final_utilization_ratio(self)
```

```
    def weld_length(self, design_dictionary):
        self.plates_width = float(design_dictionary[
            KEY_PLATE_WIDTH])
        # Use the weld_size that was already processed in
            design_of_weld
        # self.weld_size = float(design_dictionary[KEY_WELD_SIZE
            ])  # This line is no longer needed
        self.cover_plate = design_dictionary[KEY_COVER_PLATE]
        # Dictionary to store output values for UI display
        self.output_values_dict = {}
        self.material = design_dictionary[KEY_MATERIAL]
        self.fu = float(design_dictionary[
            KEY_DP_WELD_MATERIAL_G_O])
        self.weld_type = design_dictionary[KEY_DP_WELD_TYPE]
        plate1_thk = float(design_dictionary[KEY_PLATE1_THICKNESS
            ])
        plate2_thk = float(design_dictionary[KEY_PLATE2_THICKNESS
            ])

        # Calculate minimum and maximum weld sizes
        self.s_min = IS800_2007.cl_10_5_2_3_min_weld_size(
            plate1_thk, plate2_thk)
        Tmin = min(plate1_thk, plate2_thk)
        self.s_max = Tmin - 1.5

        # Check weld size constraints
        #logger.info(": Checking weld size requirements as per IS
             800:2007")
        #logger.info(": Minimum weld size required (s_min) = {}
            mm [Ref. Table 21, Cl.10.5.2.3]".format(self.s_min))
        #logger.info(": Maximum allowed weld size (s_max) = {} mm
             [Ref. Cl.10.5.3.1]".format(self.s_max))
        #logger.info(": Selected weld size = {} mm".format(self.
            weld_size))
```

```
        if self.weld_size < self.s_min or self.weld_size > self.
            s_max:
             self.design_status = False
             if self.weld_size < self.s_min:
                 logger.error(": Weld size fails: Size {} mm is
                     less than minimum required {} mm".format(
                      self.weld_size, self.s_min))
                 logger.info(": Design action required: Increase
                     the weld size to at least {} mm".format(self.
                     s_min))
             else:
                 logger.error(": Weld size fails: Size {} mm
                     exceeds maximum allowed {} mm".format(
                      self.weld_size, self.s_max))
                 logger.info(": Design action required: Decrease
                     the weld size to at most {} mm".format(self.
                     s_max))
             logger.error(": Design status: UNSAFE")
             logger.info(": =========End Of Design==========")
             return

        # Calculate weld length since size is acceptable
        if "shop weld" in self.weld_type.lower():
             self.gamma_mw = 1.25
        else:
             self.gamma_mw = 1.50

        self.f_w = self.fu / (math.sqrt(3) * self.gamma_mw)  #
            Design strength of weld

        if "single" in self.cover_plate.lower():
             self.N_f = 1  # Number of welds
        else:
```

```
            self.N_f = 2  # Double cover plate means two weld
                interfaces

                # Calculate required weld length
        self.L_req = self.tensile_force / (self.N_f * 0.707 *
            self.weld_size * self.f_w)

        # Check if straight weld is sufficient
        if self.L_req <= self.plates_width:
            logger.info(": Straight weld will be provided as
                required length is less than plate width")
            self.weld_length_provided = self.plates_width
            self.weld_length_effective = self.
                weld_length_provided
            self.weld_angle = 0
            self.side_weld_length = 0

        else:
            # Calculate skewed weld parameters
            L_target = self.L_req / self.N_f  # Required length
                per weld line

            # Calculate skew angle
            self.weld_angle = math.degrees(math.atan((L_target -
                self.plates_width)/(2 * self.plates_width)))

            # Constrain angle between 20-60 degrees
            if self.weld_angle < 20:
                self.weld_angle = 20
            elif self.weld_angle > 60:
                self.weld_angle = 60

            # Calculate provided length per weld line with skew
            L_provided_line = self.plates_width + 2 * self.
```

```
                plates_width * math.tan(math.radians(self.
                weld_angle))
            L_provided_total = self.N_f * L_provided_line

            # Check if side welds are needed
            if L_provided_total < self.L_req:
                # Calculate required side weld length
                L_side = (self.L_req - L_provided_total) / self.
                    N_f

                # Calculate minimum return weld length
                min_return = max(2 * self.weld_size, 10)  # As
                    per IS 800:2007 Cl 10.5.10.2
                L_side = max(L_side, min_return)

                self.side_weld_length = L_side
            else:
                self.side_weld_length = 0

            self.weld_length_provided = L_provided_total
            self.weld_length_effective = L_provided_total + (2 *
                self.side_weld_length * self.N_f)

            logger.info(": Skewed weld will be provided with
                angle {:.2f} degrees".format(self.weld_angle))

        # Update output values for UI display
        self.output_values_dict[KEY_OUT_WELD_LENGTH] = self.
            weld_length_effective

    def weld_strength_verification(self, design_dictionary):
        """Verify weld strength and calculate utilization"""
        logger.info(": ========== Checking Weld Strength
            ==========")
```

```
        # Use the weld_size that was already processed in
            design_of_weld
        # self.weld_size = float(design_dictionary[KEY_WELD_SIZE
            ])  # This line is no longer needed

        # Ensure we have weld_length_provided from previous
            calculation
        if not hasattr(self, 'weld_length_provided'):
            logger.error(": Weld length must be calculated before
                strength verification")
            self.design_status = False
            return

        # Calculate effective length by subtracting 2 times weld
            size from provided length
        self.weld_length_effective = self.weld_length_provided -
            (2 * self.weld_size)

        logger.info(": Checking minimum length requirements...")
        # Check if effective length meets minimum requirement of
            4 times weld size
        min_length = 4 * self.weld_size
        if self.weld_length_effective < min_length:
            self.design_status = False
            logger.error(": Effective weld length {} mm is less
                than minimum required length {} mm [Ref. Cl
                .10.5.4, IS 800:2007]".format(
                 round(self.weld_length_effective, 2), round(
                    min_length, 2)))
            logger.info(": Increase the weld length or size")
            return
        else:
            logger.info(": Minimum length requirement satisfied")
```

```
        # Calculate weld strength
        self.weld_strength = self.f_w * 0.707 * self.weld_size *
            self.weld_length_effective * self.N_f

        # Calculate weld utilization ratio
        weld_utilization = self.tensile_force / self.
            weld_strength
        self.utilization_ratios['weld'] = weld_utilization

        #logger.info(": Weld Strength Calculation Results:")
        #logger.info(": Design strength of weld (f_w) = {} N/mm
            ".format(round(self.f_w, 2)))
        #logger.info(": Effective throat thickness = {} mm".
            format(round(0.707 * self.weld_size, 2)))
        #logger.info(": Weld size = {} mm".format(self.weld_size)
            )
        #logger.info(": Effective length = {} mm".format(round(
            self.weld_length_effective, 2)))
        #logger.info(": Number of weld interfaces = {}".format(
            self.N_f))
        #logger.info(": Calculated weld strength = {} kN".format(
            round(self.weld_strength/1000, 2)))
        #logger.info(": Required tensile force = {} kN".format(
            round(self.tensile_force/1000, 2)))
        #logger.info(": Weld utilization ratio = {}".format(round
            (weld_utilization, 3)))

        if weld_utilization > 1:
            logger.error(": Weld strength is insufficient")
            logger.info(": Increase weld size or length")
        else:
            logger.info(": Weld strength is adequate")
```

```
    def long_joint_reduction_factor(self):
        """Calculate reduction factor for long joints according
           to IS 800:2007 Cl. 10.5.7.1(b)"""

        # Calculate effective throat thickness
        a = 0.707 * self.weld_size

        # Check if reduction is needed
        if self.weld_length_effective <= 150 * a:
            self.beta_L = 1.0
            logger.info(": No reduction for long joints required
               as length is less than 150 times throat thickness"
               )
            return

        # Calculate reduction factor
        self.beta_L = 1.2 - (0.2 * self.weld_length_effective)
           /(150 * a)

        # Ensure minimum value of 0.8
        self.beta_L = max(0.8, self.beta_L)

        # Adjust weld design strength and recalculate utilization
        self.f_w_adjusted = self.f_w * self.beta_L

        # Recalculate weld strength with reduction factor
        self.weld_strength_reduced = self.f_w_adjusted * 0.707 *
           self.weld_size * self.weld_length_effective * self.N_f

        # Update weld utilization with reduced strength
        weld_utilization_reduced = self.tensile_force / self.
           weld_strength_reduced
        self.utilization_ratios['weld'] =
           weld_utilization_reduced  # Update the utilization
```

```
            ratio

        #logger.info(": Long joint reduction check results:")
        #logger.info(": Long joint reduction factor  L  = {}".
            format(round(self.beta_L, 2)))
        #logger.info(": Original weld design strength = {} N/mm
            ".format(round(self.f_w, 2)))
        #logger.info(": Adjusted weld design strength = {} N/mm
            ".format(round(self.f_w_adjusted, 2)))
        #logger.info(": Updated weld utilization ratio = {}".
            format(round(weld_utilization_reduced, 3)))

    def check_base_metal_strength(self, design_dictionary):
        """Check strength of base metal according to IS 800:2007
            """

        # Extract material properties and handle material grade
            strings
        material_grade = design_dictionary[KEY_MATERIAL]
        # Extract numeric value from material grade string (e.g.
            "E 165 (Fe 290)" -> 165)
        try:
            # Extract the number after 'E' (e.g. "E 165" -> 165)
            self.fy = float(material_grade.split('(')[0].strip().
                split()[-1])

            # For custom grades, try direct conversion
            if material_grade.startswith('Custom'):
                self.fy = float(material_grade.split('_')[1])
        except (ValueError, IndexError):
            logger.error(f": Invalid material grade format: {
                material_grade}")
            self.design_status = False
            return
```

```
        self.fu = float(design_dictionary[
            KEY_DP_WELD_MATERIAL_G_O])

        # Partial safety factors
        self.gamma_m0 = 1.10  # For yielding
        self.gamma_m1 = 1.25  # For rupture

        # Calculate areas
        Tmin = min(float(design_dictionary[KEY_PLATE1_THICKNESS])
            ,
                   float(design_dictionary[KEY_PLATE2_THICKNESS])
                      )
        self.A_g = Tmin * self.plates_width
        self.A_n = self.A_g  # For welded joints, net area equals
            gross area

        # Calculate design strength based on yielding and rupture
        T_dy = self.A_g * self.fy / self.gamma_m0
        T_du = 0.9 * self.A_n * self.fu / self.gamma_m1

        # Design base metal strength is minimum of the two
        self.T_db = min(T_dy, T_du)

        # Calculate base metal utilization ratio
        base_metal_utilization = self.tensile_force/self.T_db
        self.utilization_ratios['base_metal'] =
            base_metal_utilization

        #logger.info(": Base Metal Strength Results:")
        #logger.info(": Material yield strength (fy) = {} N/mm
            ".format(round(self.fy, 2)))
        #logger.info(": Material ultimate strength (fu) = {} N/
            mm  ".format(round(self.fu, 2)))
```

```
        #logger.info(": Gross section area = {} mm ".format(
            round(self.A_g, 2)))
        #logger.info(": Net section area = {} mm".format(round(
            self.A_n, 2)))
        #logger.info(": Tensile strength - Yielding = {} kN".
            format(round(T_dy/1000, 2)))
        #logger.info(": Tensile strength - Rupture = {} kN".
            format(round(T_du/1000, 2)))
        #logger.info(": Design strength of base metal = {} kN".
            format(round(self.T_db/1000, 2)))
        #logger.info(": Required tensile force = {} kN".format(
            round(self.tensile_force/1000, 2)))
        #logger.info(": Base metal utilization ratio = {}".format
            (round(base_metal_utilization, 3)))

        if base_metal_utilization > 1:
            logger.error(": Base metal strength is insufficient [
                cl. 6.2, IS 800:2007]")
            logger.error(": Section fails in tension, try
                increasing the plate dimensions or using higher
                grade material")
        else:
            logger.info(": Base metal strength is adequate")

    def calculate_final_utilization_ratio(self):
        """Calculate final utilization ratio and set design
            status after all component checks"""
        logger.info(": ========== Final Design Check ==========
            ")

        if not hasattr(self, 'utilization_ratios'):
            logger.error(": Cannot calculate final utilization
                ratio - component checks incomplete")
            self.design_status = False
```

```
            return

        # Get maximum utilization ratio across all components
        self.utilization_ratio = max(self.utilization_ratios.
            values())

        #logger.info(": Design Status Summary:")
        #logger.info(": Weld utilization ratio: {}".format(round(
            self.utilization_ratios['weld'], 3)))
        #logger.info(": Base metal utilization ratio: {}".format(
            round(self.utilization_ratios['base_metal'], 3)))
        #logger.info(": Overall utilization ratio: {}".format(
            round(self.utilization_ratio, 3)))

        # Design is safe only if all utilization ratios are < 1.0
        if self.utilization_ratio > 1.0:
            self.design_status = False
            logger.error(": ========== Design is UNSAFE
                ==========")

            # Log which component caused the failure
            critical_component = max(self.utilization_ratios.
                items(), key=lambda x: x[1])[0]
            logger.error(": {} utilization ratio ({:.3f}) exceeds
                allowable limit of 1.0".format(
                critical_component.title(), self.
                    utilization_ratio))

            recommendations = {
                'weld': ": Consider increasing weld size or
                    length",
                'base_metal': ": Consider increasing plate
                    dimensions or using higher grade material"
            }
```

```
            logger.info(recommendations[critical_component])
        else:
            self.design_status = True
            logger.info(": ========== Design is SAFE ==========
                ")
            logger.info(": All utilization ratios are within
                acceptable limits")

        logger.info(": ==========End Of Design==========\n")

    def save_design(self, popup_summary):
        """Save design details for report generation"""

        # Report input dictionary
        self.report_input = {
            KEY_MODULE: self.module,
            KEY_MAIN_MODULE: self.mainmodule,

            # Connection details
            KEY_DISP_AXIAL: round(self.tensile_force/1000, 2),  #
                Convert N to kN

            # Connecting Members
            "Connecting Members": "TITLE",
            KEY_DISP_PLATETHK: str([int(d) for d in [self.plate1.
                thickness[0], self.plate2.thickness[0]]]),
            KEY_DISP_MATERIAL: self.main_material,
            KEY_DISP_ULTIMATE_STRENGTH_REPORT: self.plate1.fu,
            KEY_DISP_YIELD_STRENGTH_REPORT: self.plate1.fy,
            KEY_DISP_PLATE_WIDTH: self.plates_width,

            # Weld Details
            "Weld Details - Input and Design Preference": "TITLE"
                ,
```

```
            KEY_DISP_DP_WELD_TYPE: self.weld_type,
            KEY_DISP_DP_WELD_FAB: self.weld.fabrication,
            KEY_DISP_DP_WELD_MATERIAL_G_O_REPORT: self.weld.fu,
            KEY_DISP_WELD_SIZE: self.weld_size,

            # Safety Factors
            "Safety Factors": "TITLE",
            KEY_DISP_GAMMA_M0: self.gamma_m0,
            KEY_DISP_GAMMA_M1: self.gamma_m1,
            KEY_DISP_GAMMA_MW: self.gamma_mw
        }

        self.report_check = []

        # Selected Member Data
        t1 = ('Selected', 'Selected Member Data', '|p{5cm}|p{2cm
           }|p{2cm}|p{2cm}|p{4cm}|')
        self.report_check.append(t1)

        if self.design_status:
            # Member Check
            t1 = ('SubSection', 'Member Check', '|p{2.5cm}|p{4.5
               cm}|p{7.5cm}|p{1cm}|')
            self.report_check.append(t1)

            t1 = (KEY_DISP_TENSION_YIELDCAPACITY, '',
                  cl_6_2_tension_yield_capacity_member(l=None, t=
                     None, f_y=self.plate1.fy, gamma=self.
                     gamma_m0,
                                                          T_dg=round(
                                                             self.T_db
                                                             /1000, 2)
                                                             , area=
                                                             self.A_g)
```

```
                                                        , '')
        self.report_check.append(t1)

        # Weld Design
        t1 = ('SubSection', 'Weld Design', '|p{3cm}|p{6.5cm}|
           p{5cm}|p{1cm}|')
        self.report_check.append(t1)

        t1 = (DISP_MIN_WELD_SIZE,
              cl_10_5_2_3_min_fillet_weld_size_required(self.
                 weld_connecting_plates, self.weld.min_weld,
                 self.weld.red),
              display_prov(self.weld_size, "s"),
              get_pass_fail(self.weld.min_weld, self.
                 weld_size, relation="leq"))
        self.report_check.append(t1)

        t1 = (DISP_MAX_WELD_SIZE,
              cl_10_5_3_1_max_weld_size(self.
                 weld_connecting_plates, self.weld_size_max),
              display_prov(self.weld_size, "s"),
              get_pass_fail(self.weld_size, self.
                 weld_size_max, relation="leq"))
        self.report_check.append(t1)

        t1 = (DISP_THROAT,
              cl_10_5_3_1_throat_thickness_req(),
              cl_10_5_3_1_throat_thickness_weld(self.
                 weld_size, self.Kt),
              get_pass_fail(3.0, self.weld_size, relation="
                 leq"))
        self.report_check.append(t1)

        t1 = (DISP_EFF, "",
```

```
            display_prov(self.weld_length_effective, "l_w")
                , "")
        self.report_check.append(t1)

        t1 = (DISP_WELD_STRENGTH,
            weld_strength_req(V=0.0, A=self.tensile_force,
                M=0.0, Ip_w=1.0,
                            y_max=0.0, x_max=0.0, l_w=self.
                                weld_length_effective,
                            R_w=self.weld.stress),
            cl_10_5_7_1_1_weld_strength(weld_conn_plates_fu
                =[self.fu], gamma_mw=self.gamma_mw,
                                        t_t=round(self.weld.
                                            throat, 2),
                                        f_w=round(self.weld.
                                            strength, 2)),
            get_pass_fail(self.weld.stress, self.weld.
                strength, relation="leq"))
        self.report_check.append(t1)

        # Long joint check if applicable
        if hasattr(self, 'beta_L'):
            t1 = (KEY_OUT_LONG_JOINT_WELD,
                long_joint_welded_req(),
                    cl_10_5_7_3_weld_strength_post_long_joint(h
                        =self.plates_width,
                                                            l=
                                                                self
                                                                .
                                                                weld_length
                                                                ,
                                                            t_t
                                                                =
```

```
                                                    self
                                                    .
                                                    weld
                                                    .
                                                    throat
                                                    ,

                                                ws=
                                                    self
                                                    .
                                                    weld
                                                    .
                                                    strength
                                                    ,

                                                wsr
                                                    =
                                                    self
                                                    .
                                                    weld
                                                    .
                                                    strength_re
                                                    )
                                                    ,

                                                    "
                                                    "
                                                    )

            self.report_check.append(t1)

            t1 = (KEY_OUT_DISP_RED_WELD_STRENGTH,
                  display_prov(round(self.weld.stress, 2), "
                     f_w"),
```

```
                    display_prov(round(self.weld.strength_red,
                        2), "f_wd"),
                    get_pass_fail(self.weld.stress, self.weld.
                        strength_red, relation="leq"))
            self.report_check.append(t1)

        # Final Checks
        t1 = ('SubSection', 'Capacity Checks', '|p{3.5cm}|p
            {4.5cm}|p{6cm}|p{1.5cm}|')
        self.report_check.append(t1)

        t1 = ('Base Metal Strength (kN)',
              display_prov(round(self.tensile_force/1000, 2),
                  "P"),
              display_prov(round(self.T_db/1000, 2), "T_db"),
              get_pass_fail(self.tensile_force, self.T_db,
                  relation="leq"))
        self.report_check.append(t1)

        t1 = ('Overall Utilization Ratio',
              required_IR_or_utilisation_ratio(IR=1),
              display_prov(round(self.utilization_ratio, 3),
                  "IR"),
              get_pass_fail(self.utilization_ratio, 1,
                  relation="leq"))
        self.report_check.append(t1)

    else:
        t1 = ('SubSection', 'Design Status', '|p{3.5cm}|p{4.5
            cm}|p{6cm}|p{1.5cm}|')
        self.report_check.append(t1)
        t1 = ('Design Status', '', 'Design Fails', 'Fail')
        self.report_check.append(t1)
```

```
        # Images
        Disp_2d_image = []
        Disp_3D_image = "/ResourceFiles/images/3d.png"

        rel_path = os.path.abspath(".")
        rel_path = rel_path.replace("\\", "/")

        fname_no_ext = popup_summary['filename']

        CreateLatex.save_latex(CreateLatex(), self.report_input,
           self.report_check, popup_summary,
                                fname_no_ext, rel_path,
                                   Disp_2d_image, Disp_3D_image,
                                   module=self.module)
```

3.4 Task 2: Documentation

The primary documentation for this module is the **Design and Detailing Checklist (DDCL) for Welded Butt Joints** that was developed as part of Task 1. The development of the `butt_joint_welded.py` module was directly guided by this DDCL, which serves as the authoritative reference for the module's logic and calculations.

- **Ensuring Transparency:** The DDCL provides a complete, step-by-step breakdown of the engineering principles from IS 800:2007. Any user or developer can refer to the DDCL and see exactly how a specific value in the software was calculated, making the module's operation completely transparent.

- **Code Verification:** The DDCL acts as the basis for verification and validation. The implemented Python code in this module can be checked against the formulas and logical flows detailed in the checklist to ensure correctness and strict adherence to the standard.

- **Internal Documentation:** In addition to the external DDCL, the Python code itself is documented with inline comments and a structured logging system. This

provides developers with immediate context for the code's functionality and helps in debugging and future maintenance.

Together, the DDCL and the well-commented code form a comprehensive documentation package that ensures the Welded Butt Joint module is reliable, verifiable, and easy to understand.

Chapter 4

Internship Task 3: Implementation of the Welded Lap Joint Design Module

This module was collaboratively developed by Nishi Kant Mandal (`https://github.com/nishikantmandal007`) and Tanu Singh (`https://github.com/Tanu-Singh-22`) during the fellowship period.

4.1 Task 3: Problem Statement

The objective of this task was to engineer and integrate a new design module for **Welded Lap Joints** into the Osdag software. This required creating a comprehensive tool capable of automatically designing a safe and efficient lap joint connection based on user-specified inputs, such as plate dimensions, material properties, and the applied tensile force. The entire design process had to be in strict compliance with the provisions outlined in the Indian Standard IS 800:2007, particularly those in Clause 10.5 concerning fillet welds. The final module needed to perform all necessary calculations, including weld size validation, strength and length calculations, and checks for long joints and base metal failure, ultimately delivering a clear "SAFE" or "UNSAFE" verdict.

4.2 Task 3: Tasks Done

The development of the Welded Lap Joint module was a comprehensive process of translating the engineering requirements from the IS code and the pre-defined Design and

Detailing Checklist (DDCL) into a robust and reliable software component. The focus was on creating a logical, maintainable, and accurate implementation that fits seamlessly within the Osdag ecosystem.

4.2.1 Module Architecture and Integration

The initial phase focused on setting up the module's structure to align with Osdag's established framework.

- A new Python class, `LapJointWelded`, was created, inheriting from the base `MomentConnection` class. This ensures consistency with the platform's architecture and allows it to interface correctly with the main application.
- The necessary functions for defining the user interface were implemented. This included `input_values` to specify the required user inputs (plate thickness, width, etc.), `weld_values` and `detailing_values` to configure the design preference tabs, and `output_values` to structure the results display.

4.2.2 Core Design Logic Implementation

The most critical part of the task was to implement the step-by-step design process for a welded lap joint as specified by IS 800:2007. This was achieved by creating a series of dedicated methods, each handling a specific part of the design verification.

- **Input Validation and Initialization:** The `set_input_values` function was coded to receive the input dictionary from the GUI, validate the data, and initialize the necessary `Plate` and `Weld` objects that store the component properties for the two plates being joined.
- **Weld Design Orchestration:** The main design flow is controlled by the `design_of_weld` method. This function acts as an orchestrator, calling the individual check functions in the correct sequence, starting with the critical weld size validation.
- **Weld Size Validation:** A dedicated method, `weld_size_check`, was implemented to ensure the user-selected weld size is compliant with IS 800:2007. It programmatically calculates the minimum required weld size (based on the thickness of the

thicker plate as per Table 21) and the maximum allowed size (based on the thinner plate). This check serves as the first gate in the design sequence; if the selected size is invalid, the design process is halted with an appropriate error message.

- **Weld Strength Calculation:** The `calculate_weld_strength` function computes the design strength of the fillet weld per unit length. This involves first calculating the effective throat thickness (typically 0.7 times the weld size) and then determining the governing strength by taking the minimum of the weld metal strength and the parent metal strength, as specified in Clause 10.5.7.

- **Weld Length Calculation:** The `calculate_weld_length` function uses the calculated weld strength to determine the required effective length of the weld needed to resist the applied tensile force. A key part of this implementation is the subsequent check against codal limits, ensuring the required length is not less than the minimum ($l_{eff,min} = \max(4s, 40\text{mm})$) or greater than the maximum ($l_{eff,max} = 70s$).

- **Long Joint and Detailing Checks:** The `check_long_joint` method was coded to handle cases of long weld lines. It checks if the effective length exceeds 150 times the throat thickness. If it does, the function calculates the mandatory reduction factor (β_{lw}) and applies it to find the final design capacity of the connection. This method also calculates detailing parameters such as the required end return length and minimum overlap length.

- **Base Metal Verification:** The `check_base_metal_strength` function ensures the plates themselves are safe. It checks the capacity of the plate against failure from gross section yielding (Clause 6.2) and net section rupture (Clause 6.3), incorporating the appropriate shear lag factor for lap joints.

- **Final Utilization Ratio:** Finally, the `calculate_final_utilization_ratio` method aggregates the utilization ratios from all preceding checks (weld capacity and base metal capacity), identifies the highest and most critical ratio, and sets the final design status of the entire connection to either "SAFE" or "UNSAFE".

4.2.3 Output and Reporting

Clear communication of the design results was a priority.

- The `output_values` function was configured to populate the GUI's output panel with the final design parameters, such as the provided weld size, effective length, connection length, and the final utilization ratio.
- The `save_design` function was implemented to gather all input and output data and generate a detailed, professional LaTeX design report, allowing users to document and share the results.

4.3 Task 3: Python Code

The complete implementation for this module is contained within the `lap_joint_welded.py` file. The script defines the `LapJointWelded` class and all the methods required to perform the design calculations and checks.

4.3.1 Description of the Script

The script follows an object-oriented approach, encapsulating all the logic for the lap joint connection within the `LapJointWelded` class. This class manages the design process from start to finish, interacting with helper classes and utility functions from the Osdag common library to ensure consistency and adherence to IS 800:2007.

4.3.2 Code Snippets and Explanation

The following snippets highlight key parts of the implemented logic.

Listing 4.1: Weld Size Validation Logic

```
1 def weld_size_check(self, design_dictionary):
2     logger.info(": ================ Weld Size Check ================")
3     weld_size = design_dictionary[KEY_WELD_SIZE]
4     plate1_thk = float(design_dictionary[KEY_PLATE1_THICKNESS])
5     plate2_thk = float(design_dictionary[KEY_PLATE2_THICKNESS])
6     Tmin = min(plate1_thk, plate2_thk)
7     s_min = IS800_2007.cl_10_5_2_3_min_weld_size(plate1_thk, plate2_thk
          )
8     s_max = Tmin - 1.5 if Tmin >= 10 else Tmin
9
```

```
    logger.info(f": Minimum weld size required (s_min) = {s_min} mm [Ref. Table 21]")
    logger.info(f": Maximum allowed weld size (s_max) = {s_max} mm [Ref. Cl.10.5.3.1]")

    # Logic to select a valid size if 'all' is chosen or validate a single size
    # ... (omitted for brevity) ...

    if selected_size is None:
        logger.error(": Selected weld size is not suitable.")
        self.design_status = False
        return False

    self.weld_size = selected_size
    logger.info(f": Selected weld size = {self.weld_size} mm (Pass)")
    return True
```

Explanation: This function is the first critical check in the design sequence. It calculates the minimum required weld size based on the thickness of the connected plates (as per Table 21 of IS 800:2007) and the maximum permissible size. It then validates the user's selected size against these limits. If the selected size is invalid, the design process is halted immediately.

Listing 4.2: Weld Length and Long Joint Check

```
def calculate_weld_length(self):
    logger.info(": ============== Weld Length Calculation ==============")
    # Required effective weld length (Cl.10.5.4.1)
    self.weld_length_required = self.tensile_force / (2 * self.fillet_weld_design_strength)
    self.leff_min = max(4 * self.weld_size, 40)
    self.leff_max = 70 * self.weld_size

    # ... (omitted logging) ...

    if self.weld_length_required < self.leff_min:
        self.l_eff = self.leff_min
    elif self.weld_length_required > self.leff_max:
```

```
        logger.error(": Required weld length exceeds maximum allowed.")
        raise ValueError("Required weld length exceeds maximum allowed.")
    else:
        self.l_eff = self.weld_length_required

def check_long_joint(self):
    logger.info(": ============== Long Joint Check ==============")
    # IS800:2007 Cl.10.5.7.3: Long joint reduction factor
    self.beta_lw = 1.0
    if self.l_eff > 150 * self.effective_throat_thickness:
        self.beta_lw = 1.2 - 0.2 * (self.l_eff / (150 * self.effective_throat_thickness))
        self.beta_lw = max(0.6, min(self.beta_lw, 1.0))
        logger.info(f": Joint is long, reduction factor beta_lw = {self.beta_lw:.3f}")

    # ... (omitted detailing calculations) ...

    self.design_capacity = 2 * self.l_eff * self.fillet_weld_design_strength * self.beta_lw
```

Explanation: This pair of functions executes the core geometric and strength calculations for the fillet welds. The `calculate_weld_length` function first determines the required effective length, l_{eff}, needed to resist the applied load. It then validates this length against the minimum and maximum codal limits. Subsequently, the `check_long_joint` function evaluates if this length triggers the long joint provision. If the joint is classified as long, the function calculates the mandatory reduction factor, β_{lw}, which is then used to compute the final, reduced design capacity of the weld.

4.3.3 Full code

The following command imports the entire source code for the module directly into the report for complete reference. This assumes the file `lap_joint_welded.py` is placed in a `codes` sub-directory.

```
"""
Module: lap_joint_welded.py
```

```
Author: Aman, Nishi Kant Mandal, Tanu Singh
Date: 2025-07-14

Description:
    LapJointWelded is a moment connection module that represents
       a welded lap joint connection.
    It inherits from MomentConnection and follows the same
       structure and design logic as other
    connection modules (e.g., BeamCoverPlate, ColumnCoverPlate)
       used in Osdag.

Reference:
    - Osdag software guidelines and connection module structure
       documentation
"""

from .moment_connection import MomentConnection
from ...utils.common.component import *
from ...utils.common.is800_2007 import *
from ...Common import *
from ...design_report.reportGenerator_latex import CreateLatex
from ...Report_functions import *
from ...utils.common.load import Load
import logging

import math

from PyQt5.QtCore import Qt

class LapJointWelded(MomentConnection):
    def __init__(self):
        super(LapJointWelded, self).__init__()
        self.design_status = False
        self.weld_size = None
```

```
        self.weld_length_provided = None
        self.weld_strength = None
        self.weld_thickness = None
        self.plate_width = None
        self.plate_length = None
        self.plate_thickness = None
        self.weld_type = None
        self.weld_material = None
        self.weld_fabrication = None
        self.weld_angle = None
        self.weld_length_effective = None

    ################################################
    # Design Preference Functions Start
    ################################################
    def tab_list(self):
        tabs = []
        tabs.append((("Weld", TYPE_TAB_2, self.weld_values)))
        tabs.append(("Detailing", TYPE_TAB_2, self.
            detailing_values))
        return tabs

    def tab_value_changed(self):
        return []

    def edit_tabs(self):
        return []

    def input_dictionary_design_pref(self):
        design_input = []
        design_input.append(("Weld", TYPE_COMBOBOX, [
            KEY_DP_WELD_TYPE,
            KEY_DP_WELD_MATERIAL_G_O
        ]))
```

```
        design_input.append(("Detailing", TYPE_COMBOBOX, [
            KEY_DP_DETAILING_EDGE_TYPE,
        ]))
        return design_input

    def input_dictionary_without_design_pref(self):
        design_input = []
        design_input.append((None, [
            KEY_DP_WELD_TYPE,
            KEY_DP_WELD_MATERIAL_G_O,
            KEY_DP_DETAILING_EDGE_TYPE,
        ], ''))
        return design_input

    def get_values_for_design_pref(self, key, design_dictionary):
        if design_dictionary[KEY_MATERIAL] != 'Select Material':
            fu = Material(design_dictionary[KEY_MATERIAL], 41).fu
        else:
            fu = ''

        defaults = {
            KEY_DP_WELD_TYPE: "Shop weld",
            KEY_DP_WELD_MATERIAL_G_O: str(fu),
            KEY_DP_DETAILING_EDGE_TYPE: "Sheared or hand flame
               cut",
        }
        return defaults.get(key)

    def detailing_values(self, input_dictionary):
        values = {
            KEY_DP_DETAILING_EDGE_TYPE: 'Sheared or hand flame
               cut',
        }
```

```
        for key in values.keys():
            if key in input_dictionary.keys():
                values[key] = input_dictionary[key]

        detailing = []

        t1 = (KEY_DP_DETAILING_EDGE_TYPE,
           KEY_DISP_DP_DETAILING_EDGE_TYPE, TYPE_COMBOBOX,
            ['Sheared or hand flame cut', 'Rolled, machine-flame
               cut, sawn and planed'],
            values[KEY_DP_DETAILING_EDGE_TYPE])
        detailing.append(t1)

        t4 = ("textBrowser", "", TYPE_TEXT_BROWSER,
           DETAILING_DESCRIPTION_LAPJOINT, None)
        detailing.append(t4)

        return detailing

    def weld_values(self, input_dictionary):
        fu = ''
        if input_dictionary and KEY_MATERIAL in input_dictionary:
            if input_dictionary[KEY_MATERIAL] != 'Select Material
               ':
                fu = Material(input_dictionary[KEY_MATERIAL], 41)
                   .fu

        values = {
            KEY_DP_WELD_TYPE: 'Shop weld',
            KEY_DP_WELD_MATERIAL_G_O: str(fu) if fu else '410',
        }

        for key in values.keys():
            if input_dictionary and key in input_dictionary:
```

```
                values[key] = input_dictionary[key]

        weld = []

        t3 = (KEY_DP_WELD_TYPE, "Type", TYPE_COMBOBOX,
            ['Shop weld', 'Field weld'],
            values[KEY_DP_WELD_TYPE])
        weld.append(t3)

        t2 = (KEY_DP_WELD_MATERIAL_G_O, "Material Grade Overwrite
           , Fu (MPa)", TYPE_TEXTBOX,
            None,
            values[KEY_DP_WELD_MATERIAL_G_O])
        weld.append(t2)
        return weld

    def set_osdaglogger(key):
        global logger
        logger = logging.getLogger('Osdag')
        logger.setLevel(logging.DEBUG)
        handler = logging.StreamHandler()
        formatter = logging.Formatter(fmt='%(asctime)s - %(name)s
            - %(levelname)s - %(message)s', datefmt='%Y-%m-%d %H
           :%M:%S')
        handler.setFormatter(formatter)
        logger.addHandler(handler)
        handler = logging.FileHandler('logging_text.log')
        formatter = logging.Formatter(fmt='%(asctime)s - %(name)s
            - %(levelname)s - %(message)s', datefmt='%Y-%m-%d %H
           :%M:%S')
        handler.setFormatter(formatter)
        logger.addHandler(handler)
        if key is not None:
            handler = OurLog(key)
```

```
            formatter = logging.Formatter(fmt='%(asctime)s - %(
               name)s - %(levelname)s - %(message)s',
                                          datefmt='%Y-%m-%d %H:%M
                                             :%S')
            handler.setFormatter(formatter)
            logger.addHandler(handler)

    def input_value_changed(self):
        lst = []
        t8 = ([KEY_MATERIAL], KEY_MATERIAL, TYPE_CUSTOM_MATERIAL,
            self.new_material)
        lst.append(t8)
        return lst

    def input_values(self):
        options_list = []
        t16 = (KEY_MODULE, KEY_DISP_LAPJOINTWELDED, TYPE_MODULE,
           None, True, 'No Validator')
        options_list.append(t16)
        t1 = (None, DISP_TITLE_CM, TYPE_TITLE, None, True, 'No
           Validator')
        options_list.append(t1)
        t31 = (KEY_PLATE1_THICKNESS, KEY_DISP_PLATE1_THICKNESS,
           TYPE_COMBOBOX, VALUES_PLATETHK_CUSTOMIZED, True, 'Int
           Validator')
        options_list.append(t31)
        t34 = (KEY_PLATE2_THICKNESS, KEY_DISP_PLATE2_THICKNESS,
           TYPE_COMBOBOX, VALUES_PLATETHK_CUSTOMIZED, True, 'Int
           Validator')
        options_list.append(t34)
        t35 = (KEY_PLATE_WIDTH, KEY_DISP_PLATE_WIDTH,
           TYPE_TEXTBOX, None, True, 'Float Validator')
        options_list.append(t35)
        t5 = (KEY_MATERIAL, KEY_DISP_MATERIAL, TYPE_COMBOBOX,
```

```
            VALUES_MATERIAL, True, 'No Validator')
        options_list.append(t5)
        t18 = (None, DISP_TITLE_WELD, TYPE_TITLE, None, True, 'No
            Validator')
        options_list.append(t18)
        t20 = (KEY_WELD_SIZE, KEY_DISP_WELD_SIZE,
            TYPE_COMBOBOX_CUSTOMIZED, VALUES_ALL_CUSTOMIZED, True,
            'No Validator')
        options_list.append(t20)
        t6 = (None, DISP_TITLE_FSL, TYPE_TITLE, None, True, 'No
            Validator')
        options_list.append(t6)
        t17 = (KEY_TENSILE_FORCE, KEY_DISP_TENSILE_FORCE,
            TYPE_TEXTBOX, None, True, 'Int Validator')
        options_list.append(t17)
        return options_list

    def customized_input(self):
        list1 = []
        t11 = (KEY_WELD_SIZE, self.weld_size_customized)
        list1.append(t11)
        return list1

    @staticmethod
    def weld_size_customized():
        return [str(size) for size in WELD_SIZES]

    def output_values(self, flag):
        out_list = []
        t21 = (None, DISP_TITLE_WELD, TYPE_TITLE, None, True)
        out_list.append(t21)
        t22 = (KEY_OUT_UTILISATION_RATIO,
            KEY_OUT_DISP_UTILISATION_RATIO, TYPE_TEXTBOX,
                round(self.utilization_ratio, 3) if flag and
```

```
                hasattr(self, 'utilization_ratio') and self.
                utilization_ratio is not None else '', True)
        out_list.append(t22)
        t23 = (KEY_OUT_WELD_TYPE, KEY_OUT_DISP_WELD_TYPE,
           TYPE_TEXTBOX,
               "Fillet" if flag else '', True)
        out_list.append(t23)
        t24 = (KEY_OUT_WELD_SIZE, KEY_OUT_DISP_WELD_SIZE,
           TYPE_TEXTBOX,
               round(self.weld_size, 1) if flag and self.
                  weld_size is not None else '', True)
        out_list.append(t24)
        t25 = (KEY_OUT_WELD_STRENGTH,
           KEY_OUT_DISP_WELD_STRENGTH_kN, TYPE_TEXTBOX,
               round(self.weld_strength / 1000, 2) if flag and
                  hasattr(self, 'weld_strength') and self.
                  weld_strength is not None else '', True)
        out_list.append(t25)
        t26 = (KEY_OUT_WELD_LENGTH_EFF,
           KEY_OUT_DISP_WELD_LENGTH_EFF, TYPE_TEXTBOX,
               round(self.weld_length_effective, 1) if flag and
                  self.weld_length_effective is not None else '',
                   True)
        out_list.append(t26)
        t27 = (KEY_OUT_WELD_CONN_LEN, KEY_OUT_DISP_WELD_CONN_LEN,
            TYPE_TEXTBOX,
               round(self.connection_length, 1) if flag and
                  hasattr(self, 'connection_length') and self.
                  connection_length is not None else '', True)
        out_list.append(t27)
        return out_list

    def module_name(self):
        return KEY_DISP_LAPJOINTWELDED
```

```
    def call_3DColumn(self, ui, bgcolor):
        if ui.chkBxCol.isChecked():
            ui.btn3D.setChecked(Qt.Unchecked)
            ui.chkBxCol.setChecked(Qt.Unchecked)
            ui.mytabWidget.setCurrentIndex(0)
        ui.commLogicObj.display_3DModel("Column", bgcolor)

    def get_3d_components(self):
        components = []
        t1 = ('Model', self.call_3DModel)
        components.append(t1)
        t3 = ('Plate1', self.call_3DColumn)
        components.append(t3)
        t4 = ('Plate2', self.call_3DPlate)
        components.append(t4)
        return components

    def call_3DPlate(self, ui, bgcolor):
        from PyQt5.QtWidgets import QCheckBox
        from PyQt5.QtCore import Qt
        for chkbox in ui.frame.children():
            if chkbox.objectName() == 'Cover Plate':
                continue
            if isinstance(chkbox, QCheckBox):
                chkbox.setChecked(Qt.Unchecked)
        ui.commLogicObj.display_3DModel("Cover Plate", bgcolor)

    def func_for_validation(self, design_dictionary):
        all_errors = []
        self.design_status = False
        flag = False
        flag1 = False
        flag2 = False
```

```
        option_list = self.input_values(self)
        missing_fields_list = []

        for option in option_list:
            if option[2] == TYPE_TEXTBOX:
                if design_dictionary[option[0]] == '':
                    missing_fields_list.append(option[1])
                else:
                    if option[0] == KEY_PLATE_WIDTH:
                        if float(design_dictionary[option[0]]) <=
                            0.0:
                            error = "Input value(s) cannot be
                               equal or less than zero."
                            all_errors.append(error)
                        else:
                            flag1 = True
                    elif option[0] == KEY_TENSILE_FORCE:
                        if float(design_dictionary[option[0]]) <=
                            0.0:
                            error = "Input value(s) cannot be
                               equal or less than zero."
                            all_errors.append(error)
                        else:
                            flag2 = True
            else:
                pass

        if len(missing_fields_list) > 0:
            error = self.generate_missing_fields_error_string(
               self, missing_fields_list)
            all_errors.append(error)
        else:
            flag = True
```

```
        if flag and flag1 and flag2:
            self.set_input_values(self, design_dictionary)
        else:
            return all_errors

    def set_input_values(self, design_dictionary):
        design_dictionary_with_defaults = design_dictionary.copy
            ()
        if KEY_SHEAR not in design_dictionary_with_defaults:
            design_dictionary_with_defaults[KEY_SHEAR] = 0.0
        if KEY_AXIAL not in design_dictionary_with_defaults:
            design_dictionary_with_defaults[KEY_AXIAL] = 0.0
        if KEY_MOMENT not in design_dictionary_with_defaults:
            design_dictionary_with_defaults[KEY_MOMENT] = 0.0

        super(LapJointWelded, self).set_input_values(self,
            design_dictionary_with_defaults)
        self.module = design_dictionary[KEY_MODULE]
        self.mainmodule = "Lap Joint Welded Connection"
        self.main_material = design_dictionary[KEY_MATERIAL]
        self.tensile_force = float(design_dictionary[
            KEY_TENSILE_FORCE]) * 1000
        self.width = float(design_dictionary[KEY_PLATE_WIDTH])
        self.plate1 = Plate(thickness=[design_dictionary[
            KEY_PLATE1_THICKNESS]],
                            material_grade=design_dictionary[
                                KEY_MATERIAL],
                            width=design_dictionary[KEY_PLATE_WIDTH])
        self.plate2 = Plate(thickness=[design_dictionary[
            KEY_PLATE2_THICKNESS]],
                                material_grade=design_dictionary[
                                    KEY_MATERIAL],
                                width=design_dictionary[
```

```
                                KEY_PLATE_WIDTH])
        self.weld = Weld(material_g_o=design_dictionary[
            KEY_DP_WELD_MATERIAL_G_O],
                         type=design_dictionary[KEY_DP_WELD_TYPE
                             ],
                         fabrication=design_dictionary.get(
                             KEY_DP_FAB_SHOP, KEY_DP_FAB_SHOP))
        self.weld.size = design_dictionary[KEY_WELD_SIZE]
        self.design_of_weld(self, design_dictionary)

    def design_of_weld(self, design_dictionary):
        logger.info(": ========== Design of Lap Joint Welded
            Connection ==========")
        logger.info(": Design Approach: IS 800:2007 Clause 10.5")
        self.utilization_ratios = {}

        if not self.weld_size_check(self, design_dictionary):
            return

        self.calculate_weld_strength(self, design_dictionary)
        self.calculate_weld_length(self)
        self.check_long_joint(self)
        self.check_base_metal_strength(self, design_dictionary)
        self.calculate_final_utilization_ratio(self)

    def weld_size_check(self, design_dictionary):
        logger.info(": =============== Weld Size Check
            ===============")
        weld_size = design_dictionary[KEY_WELD_SIZE]
        plate1_thk = float(design_dictionary[KEY_PLATE1_THICKNESS
            ])
        plate2_thk = float(design_dictionary[KEY_PLATE2_THICKNESS
            ])
        Tmin = min(plate1_thk, plate2_thk)
```

```
        s_min = IS800_2007.cl_10_5_2_3_min_weld_size(plate1_thk,
            plate2_thk)
        s_max = Tmin - 1.5 if Tmin >= 10 else Tmin

        logger.info(f": Minimum weld size required (s_min) = {
            s_min} mm [Ref. Table 21, Cl.10.5.2.3]")
        logger.info(f": Maximum allowed weld size (s_max) = {
            s_max} mm [Ref. Cl.10.5.3.1]")

        selected_size = None
        if isinstance(weld_size, str) and weld_size.lower() == '
            all':
            valid_sizes = [s for s in ALL_WELD_SIZES if s_min <=
                s <= s_max]
            if valid_sizes:
                selected_size = float(valid_sizes[0])
        else:
            try:
                size_val = float(weld_size[0] if isinstance(
                    weld_size, list) else weld_size)
                if s_min <= size_val <= s_max:
                    selected_size = size_val
            except (ValueError, IndexError):
                pass

        if selected_size is None:
            logger.error(": Selected weld size is not suitable.")
            self.design_status = False
            return False

        self.weld_size = selected_size
        logger.info(f": Selected weld size = {self.weld_size} mm
            (Pass)")
        return True
```

```
    def calculate_weld_strength(self, design_dictionary):
        logger.info(": ============== Weld Strength Calculation
            ==============")
        # IS800:2007 Cl.10.5.3.2: Throat thickness a = K * s,
            where K depends on angle
        # For fillet welds, K = sin(  ),     = weld angle (default
             45   if not specified)
        weld_angle = design_dictionary.get('weld_angle', 45)
        if not isinstance(weld_angle, (int, float)):
            try:
                weld_angle = float(weld_angle)
            except Exception:
                weld_angle = 45
        # K = sin(angle)
        K = round(math.sin(math.radians(weld_angle)), 3)
        if K <= 0:
            logger.error(f": Invalid weld angle {weld_angle}  .
                Using default K=0.7 (45  )")
            K = 0.7
        self.effective_throat_thickness = K * self.weld_size  #
            Cl.10.5.3.2
        logger.info(f": Effective throat thickness (a) = {self.
            effective_throat_thickness:.2f} mm [Cl.10.5.3.2, K={K
            },   ={weld_angle}  ]")
        self.fu = float(design_dictionary[
            KEY_DP_WELD_MATERIAL_G_O])
        self.gamma_mw = 1.25 if design_dictionary[
            KEY_DP_WELD_TYPE] == "Shop weld" else 1.50  # Cl
            .10.5.7.1
        self.weld_design_strength = (self.fu * self.
            effective_throat_thickness) / (math.sqrt(3) * self.
            gamma_mw)  # Cl.10.5.7.1
        self.parent_design_strength = 0.6 * self.fu * self.
```

```
            effective_throat_thickness / self.gamma_mw  # Cl
            .10.5.7.2
        self.fillet_weld_design_strength = min(self.
            weld_design_strength, self.parent_design_strength)
        logger.info(f": Design strength of fillet weld = {self.
            fillet_weld_design_strength:.2f} N/mm^2 [Cl.10.5.7]")
        # Weld stress check (Cl.10.5.7):
        self.weld_stress = self.tensile_force / (2 * self.
            effective_throat_thickness * self.l_eff) if hasattr(
            self, 'l_eff') and self.l_eff else 0
        if self.weld_stress > self.fillet_weld_design_strength:
            error_msg = f"Weld stress {self.weld_stress:.2f} N/mm
                ^2 exceeds design strength {self.
                fillet_weld_design_strength:.2f} N/mm^2 [Cl
                .10.5.7]"
            logger.error(": " + error_msg)
            print("[Osdag ERROR]", error_msg)
            self.design_status = False
            self.design_error = "Weld stress exceeds design
                strength."
            return

    def calculate_weld_length(self):
        logger.info(": ============== Weld Length Calculation
            ==============")
        # Required effective weld length (Cl.10.5.4.1)
        self.weld_length_required = self.tensile_force / (2 *
            self.fillet_weld_design_strength)
        self.leff_min = max(4 * self.weld_size, 40)  # Cl
            .10.5.4.1
        self.leff_max = 70 * self.weld_size  # Cl.10.5.4.1
        logger.info(f": Required effective weld length = {self.
            weld_length_required:.2f} mm")
        logger.info(f": Minimum effective weld length = {self.
```

```
            leff_min} mm [Cl.10.5.4.1]")
        logger.info(f": Maximum effective weld length = {self.
            leff_max} mm [Cl.10.5.4.1]")
        # Check min/max
        if self.weld_length_required < self.leff_min:
            self.l_eff = self.leff_min
            logger.warning(f": Required length is less than
                minimum, using l_eff = {self.l_eff} mm [Cl
                .10.5.4.1]")
        elif self.weld_length_required > self.leff_max:
            logger.error(": Required weld length exceeds maximum
                allowed. Increase weld size. [Cl.10.5.4.1]")
            self.design_status = False
            raise ValueError("Required weld length exceeds
                maximum allowed.")
        else:
            self.l_eff = self.weld_length_required
            logger.info(": Required weld length is within limits
                (Pass)")
        # Detailing: Minimum spacing between parallel fillet
            welds (Cl.10.5.4.2)
        # Not implemented here, but should be checked in GUI or
            input validation

    def check_long_joint(self):
        logger.info(": ============== Long Joint Check
            ==============")
        # IS800:2007 Cl.10.5.7.3: Long joint reduction factor
        self.beta_lw = 1.0
        if self.l_eff > 150 * self.effective_throat_thickness:
            self.beta_lw = 1.2 - 0.2 * (self.l_eff / (150 * self.
                effective_throat_thickness))
            self.beta_lw = max(0.6, min(self.beta_lw, 1.0))
            logger.info(f": Joint is long, reduction factor
```

```
                beta_lw = {self.beta_lw:.3f} [Cl.10.5.7.3]")
        else:
            logger.info(": No reduction for long joint required (
                Pass)")
        # Modified required length
        l_req_modified = self.l_eff / self.beta_lw
        if l_req_modified < self.leff_min:
            logger.warning(f": Modified required weld length {
                l_req_modified:.2f} mm is less than minimum
                effective length {self.leff_min} mm [Cl.10.5.4.1]"
                )
            self.l_eff = self.leff_min
        elif l_req_modified > self.leff_max:
            logger.error(": Modified required weld length exceeds
                 maximum allowed. Increase weld size. [Cl
                .10.5.4.1]")
            self.design_status = False
            raise ValueError("Modified required weld length
                exceeds maximum allowed.")
        else:
            self.l_eff = l_req_modified
        # End return length (Cl.10.5.4.5): min(2*s, 12mm)
        self.end_return_length = max(2 * self.weld_size, 12)  #
            Cl.10.5.4.5
        logger.info(f": End return length = {self.
            end_return_length} mm [Cl.10.5.4.5]")
        # Overlap length (Cl.10.5.4.3): min overlap = 4*s or 40mm
            , whichever is more
        self.overlap_length = max(4 * self.weld_size, 40)
        logger.info(f": Overlap length = {self.overlap_length} mm
            [Cl.10.5.4.3]")
        self.connection_length = self.l_eff + 2 * self.
            end_return_length
        # Design capacity (Cl.10.5.7.3):
```

```
        self.design_capacity = 2 * self.l_eff * self.
            fillet_weld_design_strength * self.beta_lw
        # Weld stress check (Cl.10.5.7):
        self.weld_stress = self.tensile_force / (2 * self.
            effective_throat_thickness * self.l_eff) if self.l_eff
             else 0
        if self.weld_stress > self.fillet_weld_design_strength:
             error_msg = f"Weld stress {self.weld_stress:.2f} N/mm
                ^2 exceeds design strength {self.
                fillet_weld_design_strength:.2f} N/mm^2 [Cl
                .10.5.7]"
             logger.error(": " + error_msg)
             print("[Osdag ERROR]", error_msg)
             self.design_status = False
             self.design_error = "Weld stress exceeds design
                strength."
             return
        self.utilization_ratios['weld'] = self.tensile_force /
            self.design_capacity if self.design_capacity > 0 else
            float('inf')
        logger.info(f": Provided effective length = {self.l_eff
            :.2f} mm")
        logger.info(f": Design capacity of weld = {self.
            design_capacity/1000:.2f} kN")

    def check_base_metal_strength(self, design_dictionary):
        logger.info(": ============== Base Metal Strength Check
            ==============")
        # IS800:2007 Cl.6.2.2, 6.2.3, 6.3 (shear lag)
        Tmin = min(float(design_dictionary[KEY_PLATE1_THICKNESS])
            , float(design_dictionary[KEY_PLATE2_THICKNESS]))
        self.A_g = Tmin * self.width
        self.gamma_m0 = 1.10
        self.gamma_m1 = 1.25
```

```
        # Shear lag factor (Cl.6.3.3):
        # For lap joints, net section efficiency = 0.7 (if not
            otherwise calculated)
        shear_lag_factor = 0.7
        T_dg = self.A_g * self.plate1.fy / self.gamma_m0  # Gross
            section yielding (Cl.6.2.2)
        T_dn = 0.9 * self.A_g * self.plate1.fu * shear_lag_factor
            / self.gamma_m1  # Net section rupture (Cl.6.2.3,
            6.3.3)
        self.T_db = min(T_dg, T_dn)
        self.utilization_ratios['base_metal'] = self.
            tensile_force / self.T_db if self.T_db > 0 else float(
            'inf')
        logger.info(": Design strength of plate = {self.T_db
            /1000:.2f} kN [Cl.6.2.2, 6.2.3, 6.3.3]")

    def calculate_final_utilization_ratio(self):
        logger.info(": ============== Final Check ==============="
            )
        # Eccentricity check (IS800:2007 Cl.10.5.7.4):
        # For lap joints, if eccentricity exists, reduce design
            strength accordingly (not implemented, placeholder)
        # TODO: Implement eccentricity reduction if required
        self.utilization_ratio = max(self.utilization_ratios.
            values())
        logger.info(f": Weld utilization ratio = {self.
            utilization_ratios['weld']:.3f}")
        logger.info(f": Base metal utilization ratio = {self.
            utilization_ratios['base_metal']:.3f}")
        logger.info(f": Overall utilization ratio = {self.
            utilization_ratio:.3f}")
        if self.utilization_ratio > 1.0:
            error_msg = "Design is UNSAFE. Utilization ratio
                exceeds 1.0."
```

```
            logger.error(": " + error_msg)
            print("[Osdag ERROR]", error_msg)
            self.design_status = False
            self.design_error = "Utilization ratio exceeds 1.0.
                Design is unsafe."
            return
        else:
            logger.info(": Design is SAFE.")
            self.design_status = True
        self.weld_strength = self.design_capacity
        self.weld_length_effective = self.l_eff

    def save_design(self, popup_summary):
        self.report_input = {
            KEY_MODULE: self.module,
            KEY_MAIN_MODULE: self.mainmodule,
            KEY_DISP_AXIAL: round(self.tensile_force / 1000, 2),
            "Connecting Members": "TITLE",
            KEY_DISP_PLATETHK: str([int(d) for d in [self.plate1.
                thickness[0], self.plate2.thickness[0]]]),
            KEY_DISP_MATERIAL: self.main_material,
            KEY_DISP_ULTIMATE_STRENGTH_REPORT: self.plate1.fu,
            KEY_DISP_YIELD_STRENGTH_REPORT: self.plate1.fy,
            KEY_DISP_PLATE_WIDTH: self.width,
            "Weld Details - Input and Design Preference": "TITLE"
                ,
            KEY_DISP_DP_WELD_TYPE: self.weld.type,
            KEY_DISP_DP_WELD_FAB: self.weld.fabrication,
            KEY_DISP_DP_WELD_MATERIAL_G_O_REPORT: self.weld.fu,
            KEY_DISP_WELD_SIZE: self.weld_size,
            "Safety Factors": "TITLE",
            KEY_DISP_GAMMA_MW: self.gamma_mw,
            "Weld Angle (deg)": getattr(self, 'weld_angle', 45),
            "Effective Throat Thickness (mm)": getattr(self, '
```

```
            effective_throat_thickness', None),
        "Long Joint Reduction Factor  _lw ": getattr(self, '
            beta_lw', 1.0),
        "End Return Length (mm)": getattr(self, '
            end_return_length', None),
        "Overlap Length (mm)": getattr(self, 'overlap_length'
            , None),
        "Shear Lag Factor": 0.7,
    }
    self.report_check = []
    if self.design_status:
        t1 = ('SubSection', 'Weld Design', '|p{3cm}|p{6.5cm}|
            p{5cm}|p{1cm}|')
        self.report_check.append(t1)
        t1 = (DISP_WELD_STRENGTH,
              f"Required: {self.tensile_force / 1000:.2f} kN"
                  ,
              f"Provided: {self.design_capacity / 1000:.2f}
                  kN",
              "Pass" if self.design_capacity >= self.
                  tensile_force else "Fail")
        self.report_check.append(t1)
        t1 = ('Overall Utilization Ratio',
              "<= 1.0",
              f"{self.utilization_ratio:.3f}",
              "Pass" if self.utilization_ratio <= 1.0 else "
                  Fail")
        self.report_check.append(t1)
    else:
        t1 = ('SubSection', 'Design Status', '|p{3.5cm}|p{4.5
            cm}|p{6cm}|p{1.5cm}|')
        self.report_check.append(t1)
        t1 = ('Design Status', '', 'Design Fails', 'Fail')
        self.report_check.append(t1)
```

```
        fname_no_ext = popup_summary['filename']
        CreateLatex.save_latex(CreateLatex(), self.report_input,
            self.report_check, popup_summary,
                                fname_no_ext, os.path.abspath(".").
                                    replace("\\", "/"), [], "/
                                    ResourceFiles/images/3d.png",
                                    module=self.module)
```

4.4 Task 3: Documentation

The primary documentation for this module is the **Design and Detailing Checklist (DDCL) for Welded Lap Joints** that was developed as part of Task 1. The development of the `lap_joint_welded.py` module was directly guided by this DDCL, which serves as the authoritative reference for the module's logic and calculations.

- **Ensuring Transparency:** The DDCL provides a complete, step-by-step breakdown of the engineering principles from IS 800:2007. Any user or developer can refer to the DDCL and see exactly how a specific value in the software was calculated, making the module's operation completely transparent.
- **Code Verification:** The DDCL acts as the basis for verification and validation. The implemented Python code in this module can be checked against the formulas and logical flows detailed in the checklist to ensure correctness and strict adherence to the standard.
- **Internal Documentation:** In addition to the external DDCL, the Python code itself is documented with inline comments and a structured logging system. This provides developers with immediate context for the code's functionality and helps in debugging and future maintenance.

Together, the DDCL and the well-commented code form a comprehensive documentation package that ensures the Welded Lap Joint module is reliable, verifiable, and easy to understand.

Chapter 5

Internship Task 4: Bug Detection and Quality Assurance for Splice Connection Modules

5.1 Task 4: Problem Statement

The objective of this task was to conduct a thorough quality assurance review and bug detection analysis on two critical Osdag modules: the **Beam-to-Beam Splice with Bolted Cover Plates** and the **Beam-to-Beam Splice with Welded Cover Plates**. The goal was to perform a deep-dive into the source code to identify, document, and report any deviations from the engineering principles and formulas stipulated in IS 800:2007. Furthermore, the analysis aimed to uncover issues related to software engineering best practices, including flawed design logic, structural inefficiencies, poor error handling, and lack of code modularity. This task was essential for enhancing the accuracy, reliability, and long-term maintainability of these complex connection design modules.

5.2 Task 4: Tasks Done

The task was executed by performing a systematic and in-depth audit of the source code for each module. The methodology involved a line-by-line code review, meticulously cross-referencing the implemented formulas with the relevant clauses in IS 800:2007, and analyzing the overall design logic for potential flaws. The findings for each module were

then categorized and documented in comprehensive bug reports, providing clear and actionable feedback for the development team.

5.2.1 Analysis of the Bolted Beam Splice Module

My review of the `Beam-to-Beam Splice Cover Plate (Bolted)` module uncovered several issues, which were categorized into two main groups: direct calculation errors that affect the design's accuracy, and broader structural problems in the code that impact maintainability and compliance.

Calculation and Code Compliance Issues

These issues represent direct contradictions or misinterpretations of the IS 800:2007 standard, leading to incorrect and potentially non-compliant design results.

- **Incorrect Shear Rupture Capacity:** I identified a critical calculation error in the `shear_rupture_` function. The code incorrectly applied a factor of 0.75 to the formula, a value not present in the relevant IS 800:2007 clause. The impact of this bug is a significant and incorrect reduction in the calculated shear rupture capacity. The recommended correction was to remove this non-standard multiplier to align the function directly with the formula prescribed in Clause 8.4 of IS 800:2007.

- **Redundant Shear Capacity Reduction:** In the calculation for the web plate's shear yielding capacity, I found that an extra reduction factor of 0.6 was being applied to the output of the `shear_yielding()` function. However, this function already correctly implements the shear criteria from Clause 8.4. This "double reduction" results in an overly conservative and inaccurate shear capacity. The solution is to remove the redundant 0.6 multiplier from the function call.

- **Inconsistent Block Shear Comments:** While reviewing the block shear calculation, I noted a serious discrepancy between the code's comments and its implementation. The comments referred to the failure pattern as a "C-shape" or "U-shape," but the formula itself was clearly implemented for a "Double L-shape" pattern. This inconsistency could mislead future developers. The fix is to update the comments to accurately reflect the implemented logic.

Code Structure and Architectural Issues

This category includes problems related to software design best practices, which affect the module's maintainability and reliability.

- **Hard-coded Test Values in Production Code:** I discovered a function named `hard_values` that contained hard-coded values explicitly for testing purposes. This is poor practice and poses a risk in a production environment. The recommendation was to completely remove this method from the production class and move it to a dedicated, separate test file.
- **Missing Critical Codal Checks:** My audit revealed that two important checks from the IS code were entirely missing from the design logic:
  - A check for edge preparation requirements for plates with high ultimate strength (where $f_u \geq 540$ MPa) as per Clause 8.6.1.3.
  - An explicit check to ensure the minimum length of the cover plate is at least equal to the width of the connected member, as required by Clause 8.6.3.2.
- **Undocumented Engineering Assumptions:** I found that the logic for calculating web clearance was based on a conditional value that was not derived from a specific IS 800:2007 clause. I identified this as an engineering assumption that required clear documentation and justification within the code comments for future clarity.

5.2.2 Analysis of the Welded Beam Splice Module

The review of the `Beam-to-Beam Splice Cover Plate (Welded)` module revealed a different set of critical issues, primarily related to flawed design logic, algorithmic inefficiencies, and poor error handling.

Calculation and Code Compliance Issues

These are bugs where the implementation deviates from the IS code, leading to potentially unsafe or incorrect designs.

- **Incorrect Long Joint Calculation:** The reduction factor for a long weld joint was critically flawed. The formula was using the total length of the flange plate as the joint length, l_j, instead of the actual length of the weld itself, which is a misinterpretation of Clause 10.5.7.3.

- **Inconsistent Safety Factor:** The partial safety factor γ_{m0} was found to be hard-coded as 1.1 directly within the `shear_yielding` function. This is inconsistent with the practice of retrieving standard values from the central library, creating maintenance challenges.

- **Missing Throat Thickness Check:** The module failed to perform an explicit check to verify that the calculated effective throat thickness, t_t, was greater than the minimum requirement of 3 mm, as mandated by Clause 10.5.3.1.

Design Logic and Implementation Issues

This category covers flaws in the program's algorithms and overall implementation strategy that affect its performance and reliability.

- **Risk of Infinite Loop:** A significant software robustness issue was found in the design iteration loops. The loops were structured to continue as long as a design status flag was `False`, but they lacked any mechanism to terminate if a valid solution could not be found, creating a high risk of the application freezing.

- **Inefficient Design Iteration:** The algorithm for determining the required plate length was highly inefficient, using a large, fixed increment of 50 mm in each iteration. This approach rarely finds an optimal solution and almost always results in an oversized and uneconomical design.

- **Poor Error Handling:** In the function that selects the minimum required plate thickness, I found that if no suitable thickness could satisfy the area requirement, the function would simply return the last (inadequate) thickness it checked. This dangerous behavior passes an unsafe component to the next stage of design.

5.3 Task 4: Documentation

The culmination of this task was the creation of two formal bug reports, one for each module analyzed. These documents serve as the primary deliverable and documentation for this quality assurance effort.

- **Structured Reporting:** Each report was structured to be clear and actionable for the development team. Bugs were categorized, and each entry included a description of the issue, the specific location in the code, a summary of the "Current" incorrect implementation, and a "Correct" proposed solution based on the IS code or software engineering best practices.
- **Providing Actionable Feedback:** The purpose of these documents is to provide a clear and unambiguous roadmap for fixing the identified issues. By detailing the problem and the required solution, the reports enable developers to efficiently address the bugs, thereby improving the quality and reliability of the Osdag software.
- **Contribution to Quality Assurance:** These reports form a permanent part of the project's quality assurance records. They not only guide the immediate bug-fixing process but also serve as a reference for future development and testing, helping to prevent similar issues from arising in other modules.

Chapter 6

Conclusions

The fellowship provided a comprehensive and immersive experience in the practical application of structural engineering principles and software development methodologies. The work spanned the entire development lifecycle, from foundational research and documentation to the implementation of new features and critical quality assurance. The following sections summarize the key tasks accomplished and the skills developed during this period.

6.1 Tasks Accomplished

The tasks undertaken were diverse, covering standalone tool creation, foundational documentation, new feature development, and software quality assurance. Each task contributed significantly to the Osdag project and to my growth as an engineer.

6.2 Skills Developed

The fellowship was instrumental in cultivating a wide range of technical and professional skills, bridging the gap between academic knowledge and real-world industry application.

Table 6.1: Summary of Tasks Accomplished

Task No.	**Task Title**	**Key Accomplishments**
Screening Task	ILD Analysis Tool Development	Developed a complete, standalone Python application for analyzing simply supported beams under moving loads. This involved implementing the Influence Line Diagram (ILD) method, building an interactive GUI with PyQt6, and integrating dynamic Matplotlib plotting.
Task 1	DDCL Creation for Welded Connections	Researched and authored comprehensive Design and Detailing Checklists (DDCLs) for Welded Butt and Lap Joints. This foundational work involved interpreting IS 800:2007 to define the complete design logic, formulas, and codal checks required for software implementation.
Task 2 & 3	Osdag Module Development	Co-developed and integrated two new, fully functional design modules into the Osdag framework: the `ButtJointWelded` and `LapJointWelded` connections. This required translating the DDCLs into object-oriented Python code and implementing complex design logic, including skewed weld calculations.
Task 4	Quality Assurance & Bug Detection	Conducted an in-depth code audit of two existing Osdag modules (Bolted and Welded Beam Splices). Identified, documented, and reported critical bugs related to incorrect engineering calculations, flawed design logic, and poor software practices, providing actionable recommendations for correction.

Table 6.2: Summary of Skills Developed

Skill Category	**Specific Skill**	**Application & Context**
Technical Skills	Structural Engineering Analysis	Applied IS 800:2007 standards to practical connection design. Utilized Influence Line Diagram theory for complex moving load analysis.

Table 6.2: Summary of Skills Developed (Continued)

Skill Category	**Specific Skill**	**Application & Context**
	Advanced Python Programming	Employed Object-Oriented Programming (OOP) to develop modular, maintainable, and scalable classes for the Osdag framework.
	GUI Development	Gained proficiency in PyQt6 by designing and building a complete, interactive graphical user interface from scratch for the ILD analysis tool.
	Data Visualization	Utilized Matplotlib to generate and embed dynamic, engineering-specific plots within a desktop application, enhancing analytical insight.
	Code Auditing & Debugging	Developed a systematic approach to reviewing complex codebases, identifying logical flaws, and tracing calculation errors back to their source.
	Technical Documentation	Mastered LaTeX for creating professional, high-quality technical reports, design checklists (DDCLs), and formal bug reports.
Professional Skills	Problem Solving	Demonstrated the ability to deconstruct complex engineering problems and translate them into structured, logical software solutions.

Table 6.2: Summary of Skills Developed (Continued)

Skill Category	**Specific Skill**	**Application & Context**
	Attention to Detail	Practiced meticulous cross-referencing of software implementation against engineering codes to ensure accuracy and compliance.
	Collaboration & Teamwork	Worked effectively in a team environment to co-develop, integrate, and deliver functional software modules.

Chapter A

Appendix

A.1 Work Reports

Table A.1: Internship Work Report

Name	NISHI KANT MANDAL		
Project	Osdag		
Internship	Summer Fellowship 2025		
DATE	**DAY**	**TASK**	**Hours**
15-May-2025	Thursday	Set up Osdag on Ubuntu and tested installation.	4
16-May-2025	Friday	Explored source code and studied PyQt UI layout.	4
17-May-2025	Saturday	Started redesigning landing page UI.	5
18-May-2025	Sunday	Created UI wireframes and Figma prototypes.	5

Continued on next page

Table A.1: Internship Work Report (Continued)

DATE	**DAY**	**TASK**	**Hours**
19-May-2025	Monday	Finalized landing page and module screen mockups.	5
20-May-2025	Tuesday	Studied IS 800:2007 welded connection standards.	5
21-May-2025	Wednesday	Drafted Lap Joint DDCL structure and layout.	5
22-May-2025	Thursday	Detailed weld length clauses for Lap Joint DDCL.	5
23-May-2025	Friday	Completed Lap Joint DDCL and started Butt Joint.	5
24-May-2025	Saturday	Added skew welds and cover plate details to Butt Joint.	5
25-May-2025	Sunday	Finished Butt Joint DDCL and reviewed compliance.	4
26-May-2025	Monday	Prepared DDCLs for mentor review.	4
27-May-2025	Tuesday	Built Butt Joint Welded module structure and GUI setup.	4
28-May-2025	Wednesday	Coded input values and plate parameter initialization.	5
29-May-2025	Thursday	Implemented weld size validation and base metal checks.	4

Continued on next page

Table A.1: Internship Work Report (Continued)

DATE	**DAY**	**TASK**	**Hours**
30-May-2025	Friday	Developed weld design function and cover plate logic.	4
31-May-2025	Saturday	Added weld strength verification and joint factors.	5
01-Jun-2025	Sunday	Integrated utilization ratio logic and ran tests.	4
02-Jun-2025	Monday	Started code audit of Bolted Beam Splice module.	3
03-Jun-2025	Tuesday	Found formula bugs and logic duplications.	4
04-Jun-2025	Wednesday	Documented edge detailing and redundancy issues.	4
05-Jun-2025	Thursday	Submitted detailed bug report for Bolted Beam Splice.	3
06-Jun-2025	Friday	Audited Welded Beam Splice for iteration issues.	4
07-Jun-2025	Saturday	Identified skew logic and plate sizing bugs.	4
08-Jun-2025	Sunday	Compiled Welded Beam Splice bug report.	3
09-Jun-2025	Monday	Discussed bug reports with mentor.	3

Continued on next page

Table A.1: Internship Work Report (Continued)

DATE	**DAY**	**TASK**	**Hours**
10-Jun-2025	Tuesday	Created Lap Joint Welded module framework.	4
11-Jun-2025	Wednesday	Built input/output UI and set input values logic.	4
12-Jun-2025	Thursday	Coded weld strength and length calculations.	4
13-Jun-2025	Friday	Added long joint reduction factor check.	4
14-Jun-2025	Saturday	Implemented base metal strength and utilization logic.	4
15-Jun-2025	Sunday	Added weld size check and validation logic.	4
16-Jun-2025	Monday	Developed skew angle and side weld calculations.	4
17-Jun-2025	Tuesday	Coded weld strength functions and unit tests.	4
18-Jun-2025	Wednesday	Integrated reduction factor logic and ran tests.	4
19-Jun-2025	Thursday	Added base metal verification and shear lag handling.	4
20-Jun-2025	Friday	Refactored and modularized Lap Joint Welded logic.	4

Continued on next page

Table A.1: Internship Work Report (Continued)

DATE	**DAY**	**TASK**	**Hours**
21-Jun-2025	Saturday	Added error handling for invalid design cases.	4
22-Jun-2025	Sunday	Started result UI integration and validation.	3
23-Jun-2025	Monday	Built debug log generator and output formatting.	3
24-Jun-2025	Tuesday	Tested real cases and handled corner scenarios.	4
25-Jun-2025	Wednesday	Refined Lap Joint GUI and display elements.	4
26-Jun-2025	Thursday	Created internal documentation for class structure.	3
27-Jun-2025	Friday	Added docstrings and method usage examples.	3
28-Jun-2025	Saturday	Peer-reviewed logic with mentor and applied feedback.	4
29-Jun-2025	Sunday	Added integration test cases for Lap Joint module.	3
30-Jun-2025	Monday	Completed edge case testing and final fixes.	4
01-Jul-2025	Tuesday	Polished interface logic and code comments.	3

Continued on next page

Table A.1: Internship Work Report (Continued)

DATE	**DAY**	**TASK**	**Hours**
02-Jul-2025	Wednesday	Reviewed complete implementation with senior developer.	3
03-Jul-2025	Thursday	Applied feedback and fixed skew weld handling.	4
04-Jul-2025	Friday	Wrote final report and function summaries.	3
05-Jul-2025	Saturday	Final mentor sync and PR approval.	3
06-Jul-2025	Sunday	Re-validated code and prepared merge package.	4
07-Jul-2025	Monday	Added PR documentation and commit annotations.	4
08-Jul-2025	Tuesday	Mentor cross-verification of final version.	3
09-Jul-2025	Wednesday	Final tests for both Lap and Butt Joint modules.	3
10-Jul-2025	Thursday	Created test case summary and QA checklist.	3
11-Jul-2025	Friday	Final GUI polish and style guide compliance.	3
12-Jul-2025	Saturday	Assembled final package for Osdag integration.	3

Continued on next page

Table A.1: Internship Work Report (Continued)

DATE	**DAY**	**TASK**	**Hours**
13-Jul-2025	Sunday	Prepared PR notes, screenshots, and description.	3
14-Jul-2025	Monday	Final validations	4
15-Jul-2025	Tuesday	Completed Lap Joint and opened the PR.	4

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.