FOSSEE Autumn Semester Long Internship Report

On

Refactoring and Enhancement of Structural Steel Design Modules for OSDAG Desktop Application

Submitted by

Harsh Chelani

3rd Year B.Tech Student, Department of CSE

VIT Bhopal University

Madhya Pradesh

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

February 3, 2026

Acknowledgments

- Start with a general statement of thanks. Express your overall gratitude to everyone who supported you during your project or research.
- Project staff at the Osdag team, Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia,
- Osdag Principal Investigator (PI) Prof. Siddhartha Ghosh, Department of Civil Engineering at IIT Bombay
- FOSSEE PI Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay
- FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team
- Acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their role in facilitating this project
- Acknowledge your colleagues who worked with you during your internship or project.
- If appropriate, thank your college, department, head, and principal for their support during your studies.

Contents

1 Introduction **5**

1.1 National Mission in Education through ICT 5

1.1.1 ICT Initiatives of MoE 6

1.2 FOSSEE Project 7

1.2.1 Projects and Activities 7

1.2.2 Fellowships 7

1.3 Osdag Software 8

1.3.1 Osdag GUI 9

1.3.2 Features 9

2 Screening Task **10**

2.1 Problem Statement 10

2.2 Tasks Done 10

3 Internship Task 1: Refactoring of Tension Welded Connection Module in Osdag Desktop **12**

3.1 Task 1: Problem Statement 12

3.2 Task 1: Tasks Done 12

3.3 Task 1: Python Code 13

3.3.1 Description of the Script 13

3.3.2 CAD Rendering with Hover Interaction 13

3.3.3 Component-Specific CAD Rendering 14

3.3.4 Fallback Full Assembly Rendering 15

3.3.5 Hover Dictionary Construction 16

3.3.6 GUI Integration 17

3.4 Task 1: Documentation 18

3.4.1 Directory Structure 18

3.4.2 Module Workflow Summary 18

4 Internship Task 2: Refactoring of Flexural Members – Purlins Module in Osdag Desktop **19**

4.1 Task 2: Problem Statement 19
4.2 Task 2: Tasks Done 19
4.2.1 Refactoring of Core Design Logic 20
4.2.2 Logger and Error Handling Improvements 21
4.2.3 CAD Visualization and Hover Interaction 22
4.2.4 GUI Integration and Module Navigation 23
4.2.5 Resource and UI Updates 24
4.2.6 Testing and Verification 24
4.3 Task 2: Documentation 24
4.3.1 Directory Structure 24
4.3.2 Module Workflow Summary 25

5 Internship Task 3: Refactoring and Integration of Simple Connections Modules in Osdag Desktop 26

5.1 Task 3: Problem Statement 26
5.2 Task 3: Tasks Done 26
5.2.1 Standardization of Output and Reporting Keys 27
5.2.2 Refactoring of Simple Connection Design Logic 27
5.2.3 CAD Logic and Hover Interaction Fixes 28
5.2.4 GUI Integration and Navigation Enhancements 29
5.2.5 Testing and Validation 30
5.3 Task 3: Documentation 31
5.3.1 Directory Structure 31
5.3.2 Module Workflow Summary 31

6 Internship Task 4: Refactoring of Base Plate Connection Module in Osdag Desktop 33

6.1 Task 4: Problem Statement 33
6.2 Task 4: Tasks Done 33
6.2.1 Core Design Logic Cleanup 34
6.2.2 CAD Visualization and Hover Interaction 34
6.2.3 GUI Integration 34
6.2.4 Testing and Validation 34

6.3 Task 4: Python Code 35
6.3.1 Description of the Script 35
6.3.2 Representative Code Snippets 35
6.3.3 Explanation of the Code 38
6.4 Task 4: Documentation 38
6.4.1 Directory Structure 38
6.4.2 Module Workflow Summary 39

7 Conclusions **40**
7.1 Tasks Accomplished 40
7.2 Skills Developed 42

A Appendix **44**
A.1 Work Reports 44

Bibliography **51**

Chapter 1

Introduction

1.1 National Mission in Education through ICT

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

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

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

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

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

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

1.1.1 ICT Initiatives of MoE

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

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

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

1.2 FOSSEE Project

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

1.2.1 Projects and Activities

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

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

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

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

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

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

1.3.1 Osdag GUI

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

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

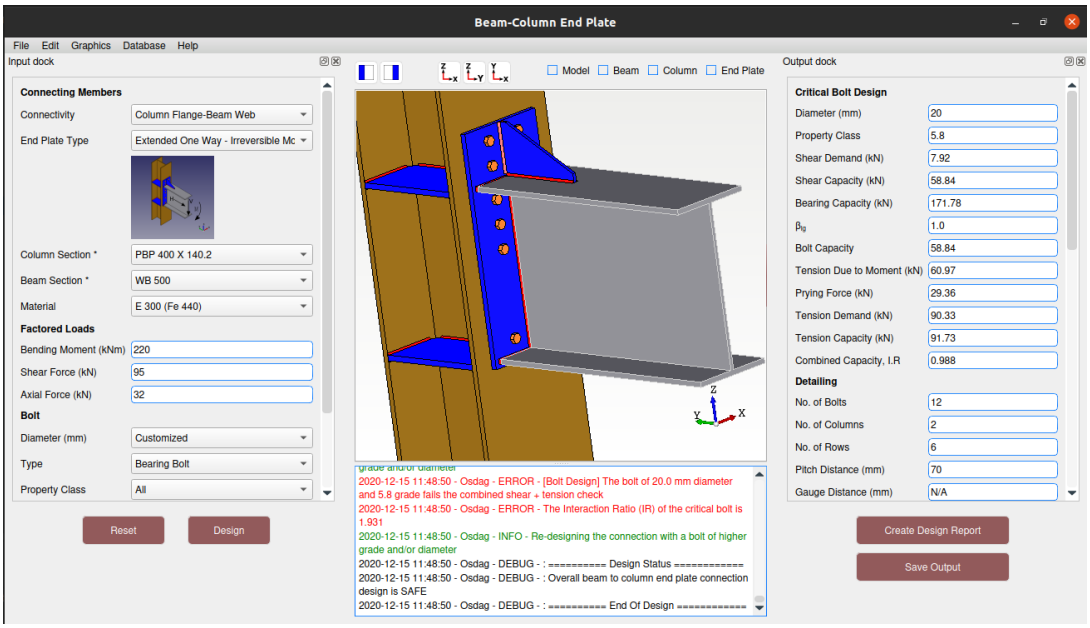

Figure 1.2: Osdag GUI

1.3.2 Features

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

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

Applicants were tasked to design and develop a modular, extensible, and containerized web-based version of the Prism Viewer application. The system aims to provide interactive three-dimensional visualization of various prism geometries through a modern web interface. The application must support multiple prism types, offer real-time user interaction through configurable controls, and maintain a clean and consistent user interface. Emphasis is placed on adopting a component-based architecture to ensure scalability, maintainability, and ease of future enhancements.

2.2 Tasks Done

The following tasks were performed as part of the screening assignment to develop the Prism Viewer web application:

- **Interactive 3D Visualization:** Implemented real-time 3D rendering of five different prism types triangular, rectangular, pentagonal, hexagonal, and octagonal using Three.js integrated with React.
- **Modular Application Architecture:** Designed a modular and extensible architecture by separating concerns into distinct components for scene rendering, user controls, and data handling.
- **Component-Based Structure:** Developed core components including:

  - `PrismViewer`: Main container component responsible for layout and state management.
  - `PrismScene`: Handles 3D scene setup, rendering, lighting, and camera controls.
  - `ControlPanel`: Provides interactive controls and displays real-time statistics.
  - `types/prism.ts`: Defines TypeScript interfaces and data models for prism configurations.

- **Interactive User Controls:** Added features such as wireframe mode, automatic rotation, orbit controls, and a statistics panel to enhance user interaction with the 3D models.

- **User Interface Design:** Implemented a clean black-and-white theme with a minimalistic design, adhering strictly to the requirement of avoiding gradients.

- **Responsive Design Implementation:** Ensured the application is fully responsive and functions seamlessly across desktop and mobile devices.

- **SEO Optimization:** Incorporated proper meta tags and semantic HTML structure to improve search engine visibility and accessibility.

- **Git Repository:** The complete source code for the Prism Viewer web application developed as part of the screening task is maintained in a public Git repository. The repository contains the full project structure, modular components, configuration files, and implementation details.

  Repository Link: `https://github.com/harshchelani08/Prism-Viewer`

Chapter 3

Internship Task 1: Refactoring of Tension Welded Connection Module in Osdag Desktop

3.1 Task 1: Problem Statement

The Tension Welded connection module in the Osdag Desktop application suffered from multiple structural and implementation issues that affected code stability, CAD visualization, and user interaction. Incorrect usage of instance references (`self`), redundant method calls, and inconsistent CAD rendering logic caused runtime errors and limited maintainability. Additionally, the absence of structured hover information and inconsistent GUI integration reduced the effectiveness of interactive CAD inspection. Hence, a systematic refactoring was required to improve code quality, maintainability, CAD interaction, and overall module robustness without altering the underlying design calculations.

3.2 Task 1: Tasks Done

This task focused on refactoring and stabilizing the Tension Welded connection module across design logic, CAD visualization, hover interaction, and GUI integration.

- Corrected improper `self`-based method invocations across design, validation, and output functions.

- Refactored CAD generation logic to correctly retrieve and render individual connection components.
- Implemented structured hover dictionaries for interactive CAD visualization.
- Fixed CAD color inconsistencies and component-wise rendering behavior.
- Integrated the module into the Osdag GUI with proper navigation and OSI file restoration support.

3.3 Task 1: Python Code

3.3.1 Description of the Script

The refactoring effort primarily targeted the CAD interaction logic implemented in `common_logic.py` and the backend Tension Welded design module. The objective was to ensure correct object binding, eliminate redundant function calls, and introduce rich hover-based interaction in the 3D CAD viewer.

3.3.2 CAD Rendering with Hover Interaction

Listing 3.1: Refactored CAD rendering logic for Tension Welded connection

```
elif self.connection == KEY_DISP_TENSION_WELDED:
    self.T = self.module_object

    # Assign hover dictionary to CAD widget
    hover_dict = self.T.hover_dict
    self.cad_widget.model_hover_labels = hover_dict

    # Create CAD assembly
    self.TObj = self.createTensionCAD()

    # Retrieve component models
    member = self.TObj.get_members_models()
    plate = self.TObj.get_plates_models()
```

```
    welds = self.TObj.get_welded_models()

    # Prepare hover labels
    label_member = ["Member", hover_dict["Member"]]
    label_plate = ["Plate", hover_dict["Plate"]]
    label_weld = ["Weld", hover_dict["Weld"]]

    # Render components
    osdag_display_shape(
        self.display, member,
        update=True, color=beam_color,
        label=label_member, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, plate,
        update=True, color=plate_color,
        label=label_plate, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, welds,
        update=True, color=weld_color,
        label=label_weld, canvas=self.cad_widget
    )
```

Explanation: This refactored logic ensures that each CAD component (member, plate, weld) is rendered individually with appropriate colors and interactive hover labels. The hover dictionary provides structured design information when users inspect the 3D model.

—

3.3.3 Component-Specific CAD Rendering

Listing 3.2: Component-wise rendering for Tension Welded connection

```
elif self.component == "Plate":
```

```
    osdag_display_shape(
        self.display, plate,
        update=True, color=plate_color,
        label=label_plate, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, welds,
        update=True, color=weld_color,
        label=label_weld, canvas=self.cad_widget
    )

elif self.component == "Endplate":
    endplate = self.TObj.get_end_plates_models()
    osdag_display_shape(
        self.display, endplate,
        update=True, color=plate_color,
        label=label_plate, canvas=self.cad_widget
    )
```

Explanation: This enhancement allows users to selectively visualize individual connection components, improving clarity during inspection and debugging of design details.

—

3.3.4 Fallback Full Assembly Rendering

Listing 3.3: Fallback rendering of complete welded connection

```
else:
    connector = BRepAlgoAPI_Fuse(welds, plate).Shape()
    shape = BRepAlgoAPI_Fuse(connector, member).Shape()

    osdag_display_shape(
        self.display, member,
        update=True, color=beam_color,
        label=label_member, canvas=self.cad_widget
```

```
    )
    osdag_display_shape(
        self.display, plate,
        update=True, color=plate_color,
        label=label_plate, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, welds,
        update=True, color=weld_color,
        label=label_weld, canvas=self.cad_widget
    )
```

Explanation: When no specific component filter is applied, all sub-components are fused and rendered together, ensuring a cohesive and complete 3D visualization of the welded connection.

—

3.3.5 Hover Dictionary Construction

Listing 3.4: Structured hover dictionary for CAD interaction

```
self.hover_dict["Member"] = (
    f"Section: {self.section_property.designation}\n"
    f"Depth: {self.section_property.depth} mm\n"
    f"Flange Thickness: {self.section_property.flange_thickness}
       mm"
)

self.hover_dict["Plate"] = (
    f"Length: {self.plate.length} mm\n"
    f"Width: {self.plate.width} mm"
)

self.hover_dict["Weld"] = (
    f"Weld Size: {self.weld_size} mm\n"
```

```
    f"Weld Strength: {self.weld_strength} N/mm"
)
```

Explanation: The hover dictionary stores detailed, component-specific design information, which is displayed interactively in the CAD viewer. This significantly improves usability and engineering clarity.

—

3.3.6 GUI Integration

Listing 3.5: GUI handler for Tension Welded connection

```
elif card_title == "Tension Welded Connection":
    self.open_tension_welded_page()

def open_tension_welded_page(self):
    title = "Tension Welded Connection"
    self.clear_layout(self.main_widget_layout)

    tension_welded = CustomWindow(title, TensionWelded, parent=
       self)

    last_design_folder = os.path.join('ResourceFiles', '
       last_designs')
    last_design_file = (
        str(tension_welded.backend.module_name()).replace(' ', ''
           ) + ".osi"
    )

    if os.path.isfile(last_design_file):
        with open(last_design_file, 'r') as last_design:
            tension_welded.setDictToUserInputs(yaml.safe_load(
               last_design))

    self.main_widget_layout.addWidget(tension_welded)
```

Explanation: This logic ensures proper navigation, tab handling, and restoration of previously saved design inputs using OSI files.

—

3.4 Task 1: Documentation

3.4.1 Directory Structure

```
src/
|-- osdag_core/
|   |-- cad/
|   |   ‘-- common_logic.py
|   ‘-- design_type/connection/
|       ‘-- tension_welded.py
|
|-- osdag_gui/
|   |-- data/
|   |   ‘-- database/
|   |       ‘-- database_config.py
|   ‘-- main_window.py
```

3.4.2 Module Workflow Summary

- User inputs are collected via the Osdag GUI.
- Design calculations are performed by the refactored backend logic.
- CAD models are generated with interactive hover-based inspection.
- Results are displayed and can be saved or exported.

Chapter 4

Internship Task 2: Refactoring of Flexural Members – Purlins Module in Osdag Desktop

4.1 Task 2: Problem Statement

The Flexural Members – Purlins module in the Osdag Desktop application suffered from multiple structural and implementation issues that affected execution stability, code maintainability, and usability. Incorrect usage of instance references (`self`), redundant method invocations, inconsistent logger handling, and improper argument passing caused runtime errors and made debugging difficult. Furthermore, the module was not fully integrated into the Osdag GUI workflow, preventing users from accessing it seamlessly from the home screen. Therefore, this task aimed to refactor the Purlins module to correct object-oriented design flaws, stabilize CAD visualization and hover interaction, improve logging, and integrate the module cleanly into the Osdag Desktop user interface without modifying the underlying design calculations.

4.2 Task 2: Tasks Done

This task focused on refactoring, stabilizing, and integrating the Flexural Members – Purlins module across the Osdag core design logic, CAD handling, logging framework, and GUI navigation.

4.2.1 Refactoring of Core Design Logic

A major part of the refactoring involved correcting the misuse of `self` and eliminating redundant object passing across multiple design functions.

Listing 4.1: Corrected section property database access

```
# Before (incorrect usage)
Flex.section_property = Flex.section_connect_database(Flex, Flex.
    result_designation)

# After (corrected usage)
Flex.section_property = Flex.section_connect_database(Flex.
    result_designation)

print(f"Web thickness: {Flex.section_property.web_thickness}")
print(f"Flange thickness: {Flex.section_property.flange_thickness
    }")
print(f"Depth: {Flex.section_property.depth}")
```

Similar corrections were applied across validation, optimization, and design routines to ensure methods operate on instance data correctly.

Listing 4.2: Corrected validation flow and input handling

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

    self.output_values(flag=False)
    option_list = self.input_values()

    missing_fields_list = []
    for option in option_list:
        if option[0] not in design_dictionary:
            missing_fields_list.append(option[1])

    if len(missing_fields_list) > 0:
```

```
        error = self.generate_missing_fields_error_string(
            missing_fields_list)
        all_errors.append(error)
        return all_errors

    self.set_input_values(design_dictionary)
    self.design(design_dictionary)
    self.results(design_dictionary)
```

—

4.2.2 Logger and Error Handling Improvements

The logging mechanism was refactored to follow a consistent, instance-based approach using Osdag's custom logger.

Listing 4.3: Standardized logger initialization

```
def set_osdaglogger(self, key):
    global logger
    logger = logging.getLogger('Osdag')
    logging.setLoggerClass(CustomLogger)

    self.logger = logging.getLogger('Osdag')
    logger.setLevel(logging.DEBUG)
    self.logger.setLevel(logging.DEBUG)

    handler = logging.StreamHandler()
    formatter = logging.Formatter(
        '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
    )
    handler.setFormatter(formatter)

    logger.addHandler(handler)
    self.logger.addHandler(handler)
```

All warning, error, and information messages were updated to use the instance logger, improving traceability during debugging.

—

4.2.3 CAD Visualization and Hover Interaction

Hover-based interaction was added to the CAD model for improved user understanding of purlin properties.

Listing 4.4: Hover dictionary population for CAD components

```
def output_values(self, flag):
    self.hover_dict["Flexural Members - Purlins"] = (
        f"<b>Purlin (C-Section)</b><br>"
        f"Length: mm<br>"
        f"Depth: mm"
    )
    return []
```

The CAD display logic was updated to attach hover labels correctly.

Listing 4.5: CAD display logic with hover labels

```
elif self.mainmodule == 'Flexural Members - Purlins':
    self.flex = self.module_object
    self.FObj = self.createPurlin()

    if self.component == "Model":
        hover_dict = self.flex.hover_dict
        self.cad_widget.model_hover_labels = hover_dict

        label_flexure = [
            "Flexural Members - Purlins",
            hover_dict["Flexural Members - Purlins"]
        ]

        osdag_display_shape(
            self.display,
```

```
            self.FObj,
            update=True,
            label=label_flexure,
            canvas=self.cad_widget
        )
```

—

4.2.4 GUI Integration and Module Navigation

The Purlins module was integrated into the Osdag GUI so that it can be launched directly from the home screen.

Listing 4.6: Module registration in database configuration

```
KEY_DISP_FLEXURE4 = ['Purlins', 'Flexural Members', '
    open_flexure_purlin']
```

A dedicated handler was added to load the module in the main window.

Listing 4.7: GUI handler for opening Purlins module

```
def open_flexure_purlin(self):
    title = "Purlin"
    self.clear_layout(self.main_widget_layout)

    flexure_purlin = CustomWindow(title, Flexure_Purlin, parent=
        self)

    last_design_folder = os.path.join('ResourceFiles', '
        last_designs')
    last_design_file = (
        str(flexure_purlin.backend.module_name()).replace(' ', ''
            ) + ".osi"
    )

    if os.path.isfile(os.path.join(last_design_folder,
        last_design_file)):
```

```
        with open(os.path.join(last_design_folder,
            last_design_file), 'r') as file:
             flexure_purlin.setDictToUserInputs(yaml.safe_load(
                file))


    self.main_widget_layout.addWidget(flexure_purlin)
```

—

4.2.5 Resource and UI Updates

The Purlins module was added to the UI card layout and application resources.

Listing 4.8: UI data update for Purlins card

```
("Purlin", ":/vectors/purlin_flexure_member.svg"),
```

Listing 4.9: Qt resource registration for Purlins SVG

```
<file>vectors/purlin_flexure_member.svg</file>
```

—

4.2.6 Testing and Verification

- Verified correct execution of design checks and optimization routines.
- Ensured CAD visualization and hover interaction functioned correctly.
- Confirmed GUI navigation and OSI-based design restoration.
- Validated that refactoring did not alter design results.

4.3 Task 2: Documentation

4.3.1 Directory Structure

```
src/
|-- osdag_core/
```

```
|   |-- cad/
|   |   `-- common_logic.py
|   |-- design_type/
|   |   `-- flexural_member/
|   |       `-- flexure_purlin.py
|
|-- osdag_gui/
|   |-- data/
|   |   |-- database/
|   |   |   `-- database_config.py
|   |   `-- ui_data.py
|   |-- resources/
|   |   |-- vectors/
|   |   |   `-- purlin_flexure_member.svg
|   |   `-- resources.qrc
|   `-- main_window.py
```

4.3.2 Module Workflow Summary

- User inputs are collected through the Osdag GUI.
- Design calculations are performed using refactored flexural logic.
- CAD models are generated with hover-enabled interaction.
- Results and reports are produced without altering original design behavior.

Chapter 5

Internship Task 3: Refactoring and Integration of Simple Connections Modules in Osdag Desktop

5.1 Task 3: Problem Statement

The Simple Connections modules in the Osdag Desktop application, including Butt Joint Bolted, Butt Joint Welded, Lap Joint Bolted, and Lap Joint Welded connections, exhibited multiple structural issues affecting stability, maintainability, and CAD interaction. Improper usage of instance references (`self`), inconsistent output dictionary keys, incorrect CAD object binding, and incomplete GUI registration caused runtime errors and limited usability.

Furthermore, reporting for simple connections lacked standardized output fields such as connection length, base metal utilization, and utilization ratio. CAD hover interaction was inconsistent across different simple connection types. Hence, a comprehensive refactoring was required to unify logic, standardize outputs, fix CAD behavior, and integrate all simple connections cleanly into the Osdag Desktop GUI.

5.2 Task 3: Tasks Done

This task focused on refactoring and enhancing all Simple Connection modules across core design logic, CAD visualization, reporting keys, and GUI integration.

5.2.1 Standardization of Output and Reporting Keys

New output keys were introduced to standardize reporting of connection length, utilization ratio, and base metal capacity for simple connections.

Listing 5.1: Addition of standardized output keys in Common.py

```
KEY_OUT_WELD_CONN_LEN = 'Weld.ConnLength'
KEY_OUT_BOLT_CONN_LEN = 'Bolt.ConnLength'
KEY_UTILIZATION_RATIO = 'UtilizationRatio'

KEY_OUT_DISP_WELD_CONN_LEN = 'Length of Connection (mm)'
KEY_OUT_DISP_BOLT_CONN_LEN = 'Length of Connection (mm)'
KEY_OUT_DISP_BASE_METAL_CAPACITY = 'Base Metal Capacity (kN)'
KEY_OUT_DISP_BASE_METAL_UTILIZATION = 'Base Metal Utilization'
```

These keys were later used uniformly across all simple connection modules and report generation logic.

—

5.2.2 Refactoring of Simple Connection Design Logic

Incorrect method calls and redundant object passing were fixed across all simple connection modules.

Butt Joint Bolted Connection

Listing 5.2: Corrected method invocation in butt_joint_bolted.py

```
# Before
if not self.check_base_metal_strength():
    self.design_status = False

# After
if not self.check_base_metal_strength(self):
    self.design_status = False
```

Butt Joint Welded Connection

Listing 5.3: Refactored utilization and connection length logic

```
self.output_dict[KEY_OUT_WELD_CONN_LEN] = self.weld_conn_length
self.output_dict[KEY_UTILIZATION_RATIO] = round(self.
    utilization_ratio, 3)
```

Lap Joint Bolted Connection

Listing 5.4: Corrected design flow in lap_joint_bolted.py

```
self.design(self, design_dictionary)
self.design(design_dictionary)

self.results(self, design_dictionary)
self.results(design_dictionary)
```

Lap Joint Welded Connection

Listing 5.5: Unified output assignment for welded lap joints

```
self.output_dict[KEY_OUT_WELD_CONN_LEN] = self.total_weld_length
self.output_dict[KEY_OUT_DISP_BASE_METAL_CAPACITY] = self.
    base_metal_capacity
```

—

5.2.3 CAD Logic and Hover Interaction Fixes

CAD generation logic was refactored to correctly bind CAD objects with their corresponding module instances.

Listing 5.6: Corrected module object reference in CAD logic

```
Col = self.module_object
print("COL_DESIGNATION :", Col.result_designation)
```

Hover dictionaries were properly linked to CAD widgets for all simple connections.

Listing 5.7: Hover dictionary assignment for CAD components

```
hover_dict = self.module_object.hover_dict
self.cad_widget.model_hover_labels = hover_dict
```

Example: Lap Joint Bolted CAD Rendering

Listing 5.8: CAD rendering with hover labels for Lap Joint Bolted

```
self.assembly, plate1, plate2, bolts, nuts = self.
   createBoltedLapJoint()

if self.component == "Model":
    osdag_display_shape(
        self.display,
        plate1,
        update=True,
        label=["Plate 1", hover_dict["plate1"]],
        canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display,
        bolts,
        update=True,
        label=["Bolts", hover_dict["bolts"]],
        canvas=self.cad_widget
    )
```

—

5.2.4 GUI Integration and Navigation Enhancements

Simple Connection modules were fully registered in the GUI database and main window.

Listing 5.9: Module registration in database_config.py

```
KEY_DISP_LAP_JOINT_BOLTED = ['Lap Joint Bolted Connection', '
   Simple Connections', 'open_lap_joint_bolted']
```

```
KEY_DISP_BUTT_JOINT_WELDED = ['Butt Joint Welded Connection', '
    Simple Connections', 'open_butt_joint_welded']
```

Listing 5.10: UI card mapping in ui_data.py

```
("Lap Joint Bolted Connection", ":/vectors/
    lap_joint_bolted_simple_connec.svg"),
("Butt Joint Welded Connection", ":/vectors/
    butt_joint_welded_simple_connec.svg"),
```

Listing 5.11: Qt resource registration for simple connection icons

```
<file>vectors/lap_joint_bolted_simple_connec.svg</file>
<file>vectors/lap_joint_welded_simple_connec.svg</file>
<file>vectors/butt_joint_bolted_simple_connec.svg</file>
<file>vectors/butt_joint_welded_simple_connec.svg</file>
```

Listing 5.12: Main window handlers for Simple Connections

```
elif card_title == "Lap Joint Bolted Connection":
    self.open_lap_joint_bolted()
elif card_title == "Lap Joint Welded Connection":
    self.open_lap_joint_welded()
elif card_title == "Butt Joint Bolted Connection":
    self.open_butt_joint_bolted()
elif card_title == "Butt Joint Welded Connection":
    self.open_butt_joint_welded()
```

—

5.2.5 Testing and Validation

- Verified correct execution of all simple connection design checks.
- Validated connection length and utilization ratio reporting.
- Confirmed CAD visualization and hover interaction for all connection types.
- Ensured GUI routing and module loading functioned correctly.

5.3 Task 3: Documentation

5.3.1 Directory Structure

```
src/
|-- osdag_core/
|   |-- Common.py
|   |-- cad/
|   |   `-- common_logic.py
|   `-- design_type/
|       `-- connection/
|           |-- lap_joint_bolted.py
|           |-- lap_joint_welded.py
|           |-- butt_joint_bolted.py
|           `-- butt_joint_welded.py
|
|-- osdag_gui/
|   |-- data/
|   |   |-- database/
|   |   |   `-- database_config.py
|   |   `-- ui_data.py
|   |-- resources/
|   |   |-- vectors/
|   |   |   |-- lap_joint_bolted_simple_connec.svg
|   |   |   |-- lap_joint_welded_simple_connec.svg
|   |   |   |-- butt_joint_bolted_simple_connec.svg
|   |   |   `-- butt_joint_welded_simple_connec.svg
|   |   `-- resources.qrc
|   `-- main_window.py
```

5.3.2 Module Workflow Summary

- User inputs are collected through the Osdag GUI for simple connections.

- Refactored core logic computes connection strength and utilization.
- CAD models are generated with component-level hover interaction.
- Standardized outputs are passed to report generation.

Chapter 6

Internship Task 4: Refactoring of Base Plate Connection Module in Osdag Desktop

6.1 Task 4: Problem Statement

The Base Plate Connection module in the Osdag Desktop application exhibited multiple logical and implementation issues that affected the stability of its CAD display, hover interaction, and GUI integration. Specifically, the CAD generation logic did not correctly incorporate all connection components (such as column, plate, weld, bolt–nut arrays, concrete, and grout models), and the hover labels were not being applied consistently. Additionally, GUI registration and routing were incomplete, leading to navigation issues in the Osdag home interface. Therefore, a comprehensive refactor was required to unify the CAD handling logic, ensure correct interaction behavior, standardize output mapping, and integrate the Base Plate module into the desktop UI without affecting design calculation accuracy.

6.2 Task 4: Tasks Done

This task focused on updating the Base Plate Connection module across core logic, CAD visualization, hover interaction, output labeling, and GUI integration.

6.2.1 Core Design Logic Cleanup

- Reviewed and refactored the `base_plate_connection.py` design logic to correct instance usage and remove redundant calls.
- Standardized connection component access functions to correctly return CAD models for all sub-components (column, plate, weld, bolt–nut array, concrete, grout).
- Improved output dictionary consistency for reporting base plate connection results.

6.2.2 CAD Visualization and Hover Interaction

- Refactored CAD display logic in `common_logic.py` to correctly display all Base Plate components with appropriate colors and labels.
- Populated the hover dictionary with detailed component names for interactive CAD tooltips.
- Enabled robust component color mapping and label binding for user clarity in the 3D viewer.

6.2.3 GUI Integration

- Registered the Base Plate Connection module in GUI configuration to allow selection from the home page.
- Updated card titles, handler functions, and navigation logic in `main_window.py` to correctly open the module.
- Integrated support for restoring previous design inputs using OSI files in the Base Plate workflow.

6.2.4 Testing and Validation

- Verified that the Base Plate CAD model renders correctly for all connection components.
- Ensured that hover labels display accurate information for plate, weld, bolts, concrete, and grout.

- Confirmed that GUI routing opens the Base Plate module without errors from the Osdag home screen.
- Performed regression verification to ensure no change in design outputs from prior versions.

6.3 Task 4: Python Code

6.3.1 Description of the Script

The refactor concentrated on CAD interaction logic within `common_logic.py` and correct mapping of multiple connection sub-components. It ensured that each component is retrieved and displayed with distinct metadata and hover labels. GUI integration included handling in `main_window.py` and the database configuration.

6.3.2 Representative Code Snippets

Listing 6.1: CAD Display Logic for Base Plate Components

```
# CAD rendering logic updated in common_logic.py
column = self.BPObj.get_column_model()
plate = self.BPObj.get_plate_connector_models()
weld = self.BPObj.get_welded_models()
nut_bolt = self.BPObj.get_nut_bolt_array_models()
conc = self.BPObj.get_concrete_models()
grout = self.BPObj.get_grout_models()

hover_dict = self.Bp.hover_dict
self.cad_widget.model_hover_labels = hover_dict.copy()

label_column = ["Column", hover_dict.get("Column")]
label_plate = ["Plate", hover_dict.get("Plate")]
label_weld = ["Weld", hover_dict.get("Weld")]
label_bolt = ["Bolt", hover_dict.get("Bolt")]
label_conc = ["Conc", hover_dict.get("Conc")]
```

```
label_grout = ["Grout", hover_dict.get("Grout")]

if self.component == "Model":
    osdag_display_shape(
        self.display, column, update=True, color=column_color,
        label=label_column, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, plate, update=True, color=plate_color,
        label=label_plate, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, weld, update=True, color=weld_color,
        label=label_weld, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, nut_bolt, update=True, color=bolt_color,
        label=label_bolt, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, conc, transparency=0.5, color=GRAY,
        update=True, label=label_conc, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, grout, transparency=0.5, color=GRAY,
        update=True, label=label_grout, canvas=self.cad_widget
    )
```

Listing 6.2: Conditional Component Rendering in Base Plate CAD

```
elif self.component == "Column":
    osdag_display_shape(
        self.display, column, update=True,
        color=column_color, label=label_column,
        canvas=self.cad_widget
```

```
    )
elif self.component == "Connector":
    osdag_display_shape(
        self.display, plate, update=True,
        color=plate_color, label=label_plate,
        canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, weld, color=weld_color,
        label=label_weld, canvas=self.cad_widget
    )
    osdag_display_shape(
        self.display, nut_bolt, update=True,
        color=Quantity_NOC_SADDLEBROWN,
        label=label_bolt, canvas=self.cad_widget
    )
```

Listing 6.3: Module Registration in GUI – main_window.py

```
# Handler mapping for Base Plate module
elif card_title == "Base Plate Connection":
    self.open_base_plate_connection()

# Opening the Base Plate design tab
def open_base_plate_connection(self):
    title = "Base Plate Connection"
    self.clear_layout(self.main_widget_layout)
    base_plate = CustomWindow(title, BasePlateConnection, parent=
       self)

    last_design_folder = os.path.join('ResourceFiles', '
       last_designs')
    last_design_file = (
        str(base_plate.backend.module_name()).replace(' ', '') +
           ".osi"
```

```
    )

    if os.path.isfile(last_design_file):
        with open(last_design_file, 'r') as last_design:
            base_plate.setDictToUserInputs(yaml.safe_load(
                last_design))

    self.main_widget_layout.addWidget(base_plate)
```

6.3.3 Explanation of the Code

- The **CAD Display Logic** snippet shows how all Base Plate connection models (column, plate, weld, bolt–nut arrays, concrete, grout) are retrieved from the backend module and displayed with separate color and hover label settings.
- The **Conditional Rendering** snippet ensures that when the viewer is filtered by component (e.g., "Column" or "Connector"), only relevant CAD models are displayed.
- The **GUI Handler** shows how the Base Plate Connection module is registered in the Osdag Desktop interface and how previous design inputs are restored via OSI files.

6.4 Task 4: Documentation

6.4.1 Directory Structure

The Base Plate Connection refactor introduced updates to both core and GUI directories:

```
src/
|-- osdag_core/
|   |-- cad/
|   |   `-- common_logic.py
|   `-- design_type/connection/
|       `-- base_plate_connection.py
```

```
|
|-- osdag_gui/
|   |-- data/
|   |   |-- database/
|   |   |   `-- database_config.py
|   |-- main_window.py
|   |-- resources/
|       `-- resources_rc.py
|   `-- win.py
```

6.4.2 Module Workflow Summary

- User selects the Base Plate Connection from the Osdag home screen.
- Input values are collected and passed to the core design engine.
- CAD models for individual components are generated and displayed.
- Hover interaction provides component details during 3D navigation.
- Results are presented and can be saved or exported.

Chapter 7

Conclusions

7.1 Tasks Accomplished

During the course of this internship, significant contributions were made toward improving, refactoring, and extending the Open Steel Design and Graphics (Osdag) software across multiple structural design modules. The work spanned core design logic, CAD visualization, GUI integration, and workflow stabilization, with a strong emphasis on code quality, maintainability, and user interaction.

The major tasks accomplished during the internship are summarized below:

- **Refactoring of the Tension Welded Member Module (Task 1):**
  - Corrected extensive `self`-related invocation errors across the design logic, validation functions, and CAD interaction layers.
  - Refactored the tension welded module to ensure consistent method signatures and clean superclass method calls.
  - Implemented structured hover dictionaries for CAD components (Member, Plate, Weld) to enable interactive 3D visualization.
  - Fixed CAD color inconsistencies and component rendering issues for individual and combined model views.
  - Integrated the Tension Welded module properly into the Osdag desktop GUI, including database configuration, navigation, and OSI file restoration.
- **Refactoring of the Flexural Members – Purlins Module (Task 2):**

- Eliminated redundant and incorrect `self`-based calls throughout the flexure purlin design logic, significantly improving execution stability.
- Refactored design workflows such as section classification, optimization checks, bending strength evaluation, web buckling checks, and result aggregation.
- Enhanced CAD rendering logic to correctly generate and display purlin models with proper camera orientation and hover-based interaction.
- Implemented hover dictionaries for flexural members to display key section properties during CAD interaction.
- Integrated the Purlins module into the Osdag GUI by updating database mappings, UI resources, card handlers, and navigation logic in the main window.

- **Stabilization and Enhancement of Design and CAD Interaction Logic (Task 3):**
  - Improved cross-module CAD logic to ensure consistent behavior for component selection, combined model rendering, and hover interaction.
  - Refined the handling of design outputs, result dictionaries, and reporting inputs to maintain alignment between backend computations and GUI visualization.
  - Addressed edge cases in design validation and output workflows to prevent runtime errors during complex design scenarios.

- **Refactoring of the Base Plate Connection Module (Task 4):**
  - Refactored the Base Plate Connection CAD logic to correctly generate and render all sub-components, including column, base plate, welds, anchor bolts, concrete, and grout models.
  - Implemented detailed hover labels for each base plate component to enhance clarity and user understanding in the 3D CAD view.
  - Unified CAD color schemes and transparency handling to ensure clear visual distinction between structural and non-structural components.
  - Integrated the Base Plate Connection module into the Osdag desktop GUI, enabling seamless access from the home screen and proper OSI file handling.

- Ensured backward compatibility and regression safety by validating design outputs against previous versions.

Overall, these tasks substantially improved the robustness, usability, and maintainability of Osdag by strengthening its design logic, CAD visualization framework, and GUI integration across multiple structural modules.

7.2 Skills Developed

The internship provided comprehensive exposure to both structural engineering concepts and advanced software development practices. The key skills and competencies developed during this period are outlined below:

- Strong understanding of **structural steel member and connection design** principles as per IS 800:2007 and related standards.
- Practical experience in translating complex engineering design procedures into **modular, maintainable Python code**.
- In-depth exposure to **large-scale code refactoring**, including removal of legacy errors, normalization of method calls, and improvement of execution flow.
- Hands-on experience with **CAD model generation and interaction** using OpenCascade-based workflows within a desktop engineering application.
- Development of **interactive CAD features** such as hover labels, component filtering, and combined model visualization.
- Experience working with **GUI frameworks and application navigation**, including module registration, UI resource management, and workflow restoration.
- Improved ability to **debug and validate engineering software**, ensuring correctness of numerical results alongside visual representation.
- Familiarity with **open-source engineering software development**, version control, and commit-based collaborative workflows.

- Enhanced skills in **technical documentation**, LaTeX-based reporting, and structured presentation of software engineering work.
- Development of professional competencies such as analytical thinking, systematic problem-solving, and writing code that balances engineering accuracy with software quality.

In conclusion, the internship provided a strong foundation in engineering software development by combining domain knowledge in structural engineering with real-world experience in refactoring, CAD integration, and GUI-driven application design. The skills and contributions developed during this period are directly applicable to future work in structural engineering software development, research, and advanced computational engineering projects.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report			
Name:		Harsh Chelani	
Project:		Osdag	
Internship:		FOSSEE Semester Long Internship (Autumn) 2025	
DATE	**DAY**	**TASK**	**Hours Worked**
01-Oct-2025	Wednesday	Installed OSDAG on Windows 11 x64 machine for desktop version and did initial testing	4
02-Oct-2025	Thursday	Installed OSDAG on Windows 11 x64 machine for web version and did initial testing	4
03-Oct-2025	Friday	Continued testing OSDAG on Windows 11 machine and validated installation process	5
04-Oct-2025	Saturday	Final testing the environment setup and fixing pip errors occurred during setup.	
06-Oct-2025	Monday	Analyzed limitations of direct browser visualization for .brep geometry	4
07-Oct-2025	Tuesday	Researched web-based CAD visualization approaches compatible with Three.js	5
08-Oct-2025	Wednesday	Exam Break	0
09-Oct-2025	Thursday	Exam Break	0
10-Oct-2025	Friday	Exam Break	0
11-Oct-2025	Saturday	Exam Break	
13-Oct-2025	Monday	Exam Break	0
14-Oct-2025	Tuesday	Explored conversion requirements for .brep to polygon-based formats	6
15-Oct-2025	Wednesday	Studied brep → obj conversion techniques and tooling options	5

16-Oct-2025	Thursday	Studied.brep → glb conversion pipeline for web rendering	6
17-Oct-2025	Friday	Compared .obj vs .glb formats for performance and browser compatibility	5
18-Oct-2025	Saturday	Presented the ideas to mentor for conversion from brep to obj	
20-Oct-2025	Monday	Designed high-level workflow for CAD conversion integration into Osdag	6
21-Oct-2025	Tuesday	Documented planned conversion pipeline and visualization approach	5
22-Oct-2025	Wednesday	Began refactoring tension member module – bolted to end gusset connection	5
23-Oct-2025	Thursday	Continued refactoring logic and validation for bolted end gusset connection	5
24-Oct-2025	Friday	Refactored tension member module – welded to end gusset connection	7
25-Oct-2025	Saturday	Created a PR to Suhail to verify the refactoring	
27-Oct-2025	Monday	Code cleanup and validation of tension member welded connection	6
28-Oct-2025	Tuesday	Regression testing of refactored tension member module	7
29-Oct-2025	Wednesday	Documentation of changes made in tension member module	6
30-Oct-2025	Thursday	Started refactoring lap joint bolted connection module	7
31-Oct-2025	Friday	Completed lap joint bolted connection refactoring	6
01-Nov-2025	Saturday	Created a PR to Suhail to verify the refactoring and recommend changes	
03-Nov-2025	Monday	Validation and testing of lap joint bolted connection	6
04-Nov-2025	Tuesday	Refactored lap joint welded connection module	5
05-Nov-2025	Wednesday	Verified lap joint welded connection design outputs	6

Internship Work Report			
Name:		Harsh Chelani	
Project:		Osdag	
Internship:		FOSSEE Semester Long Internship (Autumn) 2025	
06-Nov-2025	Thursday	Refactored butt joint bolted connection module	7
07-Nov-2025	Friday	Tested butt joint bolted connection with multiple input cases	6
08-Nov-2025	Saturday	Checked Osi file loading for different modules	
10-Nov-2025	Monday	Refactored butt joint welded connection module	7
11-Nov-2025	Tuesday	Verified butt joint welded module calculations	8
12-Nov-2025	Wednesday	Refactored flexural member purlin module	8
13-Nov-2025	Thursday	Optimized purlin module logic and code structure	6
14-Nov-2025	Friday	Validation and regression testing of purlin module	7
15-Nov-2025	Saturday	Created a PR to Suhail for Flexural members – Purlin module for him to verify the refactoring and recommend changes	
17-Nov-2025	Monday	Exam Break	0
18-Nov-2025	Tuesday	Exam Break	0
19-Nov-2025	Wednesday	Exam Break	0
20-Nov-2025	Thursday	Started refactoring base plate connection module (large desktop module)	6

21-Nov-2025	Friday	Continued refactoring base plate connection calculation logic	7
22-Nov-2025	Saturday	Matched old OSDAG UI for base plate to new one and change the colour specifications of model to new OSDAG specifications	
24-Nov-2025	Monday	Improved code structure and readability of base plate module	5
25-Nov-2025	Tuesday	Debugged edge cases in base plate connection module	6
26-Nov-2025	Wednesday	Completed refactoring of base plate module (~8000 LOC)	7
27-Nov-2025	Thursday	Performed regression testing on base plate module	6
28-Nov-2025	Friday	Documentation of base plate module refactoring	7
29-Nov-2025	Saturday	Created a PR to Suhail for Base Plate Connection module for him to verify the refactoring and recommend changes	
01-Dec-2025	Monday	Testing Header Plate / End Plate connection using random input values	7
02-Dec-2025	Tuesday	Testing Header Plate / End Plate connection using OSI input values	6
03-Dec-2025	Wednesday	Testing Beam-to-Beam Cover Plate Welded connection with random inputs	7
04-Dec-2025	Thursday	Testing Beam-to-Beam Cover Plate Welded connection using OSI inputs	6
05-Dec-2025	Friday	Verification of test results and comparison with expected outputs	5
06-Dec-2025	Saturday	Reported #ISSUE64 to Suhail – For Tension Welded Module	

Internship Work Report

Name:	Harsh Chelani
Project:	Osdag
Internship:	FOSSEE Semester Long Internship (Autumn) 2025

08-Dec-2025	Monday	Identified inconsistencies and documented testing observations	7
09-Dec-2025	Tuesday	Raised GitHub issue for identified bug (Issue #16)	6
10-Dec-2025	Wednesday	Assisted in debugging reported issues	7
11-Dec-2025	Thursday	Retesting modules after fixes	8
12-Dec-2025	Friday	Prepared detailed testing report for mentor review	8
13-Dec-2025	Saturday	Filled the form for testing and reported issues/inconsistencies In Osdag desktop version	
15-Dec-2025	Monday	Compiled documentation for .brep conversion research	6
16-Dec-2025	Tuesday	Drafted CAD visualization integration proposal for Osdag Web	7
17-Dec-2025	Wednesday	Organized internship work logs and technical notes	6
18-Dec-2025	Thursday	Started drafting final internship report	6
19-Dec-2025	Friday	Continued writing and formatting internship report	7
20-Dec-2025	Saturday	Edited work report PDF	
22-Dec-2025	Monday	Incorporated mentor feedback and refined report	6
23-Dec-2025	Tuesday	Final proofreading and technical verification of report	7
24-Dec-2025	Wednesday	Prepared final version of internship report	5
25-Dec-2025	Thursday	Helped new interns in setting up the environment	6
26-Dec-2025	Friday	Tackled an open issue and tried to Debug it	7
27-Dec-2025	Saturday	Continued tackling the open issue but wasn’t able to resolve it	
29-Dec-2025	Monday	Final report additions	6

30-Dec-2025	Tuesday	Submitted report draft for mentor to review	5
31-Dec-2025	Wednesday	Final report Submission	6

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.