FOSSEE Winter Internship Report

On

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

Submitted by

Yugh Juneja

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 8, 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

Chapter 1

Introduction

1.1 National Mission in Education through ICT

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

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

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

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

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

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

1.1.1 ICT Initiatives of MoE

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

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

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

1.2 FOSSEE Project

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

1.2.1 Projects and Activities

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

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

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

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

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

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

1.3.1 Osdag GUI

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

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

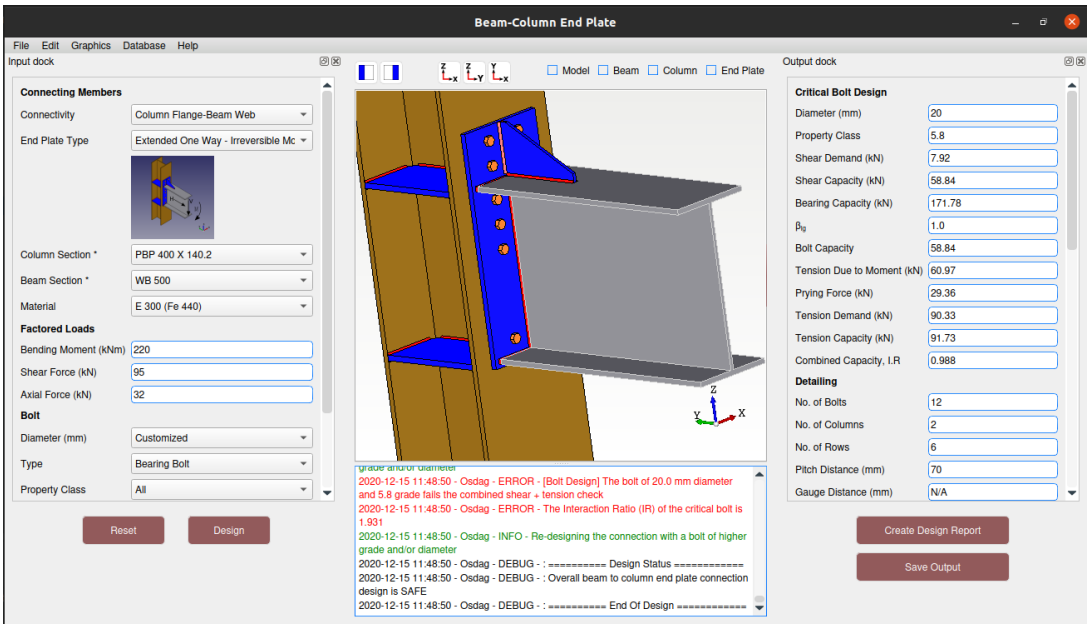


Figure 1.2: Osdag GUI

1.3.2 Features

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

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

The screening task was designed to evaluate the candidate's ability to design and develop a modern, modular, and extensible web-based application with interactive three-dimensional visualization capabilities. The objective of the task was to conceptualize and implement a web-based system capable of visualizing geometric prism shapes in real time, while allowing users to dynamically modify input parameters and observe immediate visual and numerical feedback.

The application was required to support multiple prism geometries, provide an interactive user interface, and follow a component-based software architecture to ensure scalability, maintainability, and ease of future enhancement. Emphasis was placed on clean user interface design, responsiveness across devices, and the use of modern web technologies for 3D rendering and application deployment.

2.2 Tasks Done

As part of the screening assignment, a complete web-based application named **Prism Viewer** was designed and developed. The key tasks performed during this assignment are summarized below.

2.2.1 Design and Architecture

A modular and extensible application architecture was designed using a clear separation of concerns between frontend visualization, backend computation, and data management. The system was structured to allow new geometric shapes and calculation logic to be added through a plugin-based mechanism without modifying the core application. This approach ensures long-term scalability and ease of maintenance.

2.2.2 Interactive 3D Visualization

Real-time three-dimensional visualization of prism geometries was implemented using modern WebGL-based rendering libraries. The application supports multiple prism types, including triangular, rectangular, pentagonal, hexagonal, and octagonal prisms. Users can interact with the 3D models through rotation, zoom, and orbit controls, enabling intuitive exploration of geometric shapes directly within the browser.

2.2.3 Dynamic User Interaction

The application allows users to dynamically adjust shape parameters such as dimensions and instantly observe corresponding changes in both geometry and calculated properties. Additional interactive features such as wireframe mode, automatic rotation, and a live statistics panel were implemented to enhance usability and user engagement.

2.2.4 Mathematical Computation and History Tracking

For each prism geometry, the application performs mathematical computations such as volume, surface area, and related geometric properties. All calculations are performed in real time and are displayed alongside the visual model. A calculation history feature was implemented to allow users to track previous computations and export results for reference.

2.2.5 User Interface and Responsiveness

A clean and minimalistic black-and-white user interface was implemented, strictly adhering to the requirement of avoiding gradients and unnecessary visual clutter. The

application layout was designed to be fully responsive, ensuring seamless usability across desktop, tablet, and mobile devices.

2.2.6 Backend Development and API Integration

A RESTful backend architecture was implemented to manage shape definitions, computation requests, and calculation history. The backend was designed to efficiently handle requests and provide structured responses to the frontend. Database integration was used to store calculation data, enabling persistence and retrieval of previous results.

2.2.7 Containerization and Deployment

The complete application was containerized using Docker to ensure consistency across development and deployment environments. Docker Compose was used to orchestrate frontend, backend, and database services, allowing the application to be deployed with minimal setup effort.

2.2.8 Testing and Validation

Comprehensive testing was performed to verify correctness of geometric calculations, stability of the 3D rendering pipeline, and robustness of user interactions. The application was tested across different browsers and screen sizes to ensure consistent behavior and performance.

2.2.9 Version Control and Repository

The complete source code for the Prism Viewer application was maintained in a public GitHub repository. The repository contains the full project structure, modular components, configuration files, and documentation related to the screening task.

Repository Link:

`https://github.com/yugh88/Prism-Viewer`

The successful completion of this screening task demonstrated proficiency in modern web development, software architecture design, and interactive 3D visualization, and served as the basis for selection into the semester-long internship.

Chapter 3

Internship Task 1: Refactoring of Compression Member Module for Axially Loaded Columns in Osdag Desktop

3.1 Task 1: Problem Statement

The Compression Member module for axially loaded columns in the Osdag Desktop application required refactoring due to issues related to code readability, maintainability, and improper object-oriented design practices. Redundant function calls, inconsistent instance handling, and tightly coupled logic made the module difficult to debug and extend.

The objective of this task was to refactor the column design module to improve code structure, ensure consistent execution of axial load design checks as per IS 800:2007, and integrate the module cleanly with the Osdag Desktop graphical user interface.

3.2 Task 1: Tasks Done

The following tasks were performed during this activity:

- Refactored the compression member design logic for axially loaded columns.
- Corrected improper usage of instance references and removed redundant calls.

- Improved validation flow for section properties, effective length, and axial capacity checks.
- Integrated the column design module with the Osdag Desktop GUI.
- Enabled restoration of previous design inputs using OSI files.

3.3 Task 1: Python Code

This section presents a Python implementation related to the integration of the Column Design module within the Osdag Desktop application. The code handles GUI navigation, module loading, and restoration of previously saved design inputs.

3.3.1 Description of the Script

The script is responsible for opening the Column Design module from the Osdag main window. It initializes the required backend objects, clears the existing GUI layout, and loads previously saved design inputs (OSI files) if available. This ensures continuity of user workflows and improves usability.

3.3.2 Python Code

A simplified version of the column design opening function is shown below:

Listing 3.1: Opening Column Design Module in Osdag Desktop

```
def open_column_design(self):
    """Opens the Column Design module."""
    title = "Column Design"
    self.clear_layout(self.main_widget_layout)

    column_design = CustomWindow(title, ColumnDesign, parent=self
        )

    last_design_folder = os.path.join('ResourceFiles', '
        last_designs')
```

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

    if os.path.isfile(last_design_file):
        with open(last_design_file, 'r') as file:
            design_data = yaml.safe_load(file)
            column_design.setDictToUserInputs(design_data)

    self.main_widget_layout.addWidget(column_design)
```

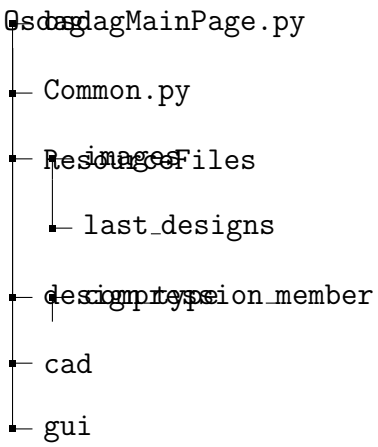
3.3.3 Explanation of the Code

- The GUI layout is cleared before loading the Column Design module.
- A `CustomWindow` instance is created for the column design backend.
- The last saved OSI file is located and loaded if it exists.
- Previous user inputs are restored to ensure workflow continuity.

3.4 Task 1: Documentation

As part of this task, contributions were made towards improving the documentation related to the Column Design module in the Osdag Developer and User Manual.

3.4.1 Directory Structure



Program Start Calls

The main entry point of the Osdag Desktop application is `osdagMainPage.py`. The application can be launched by executing the following command from the project root directory:

```
$ python osdagMainPage.py
```

Chapter 4

Internship Task 2: Development of Axially Loaded Column Module for Osdag Web

4.1 Task 2: Problem Statement

The Osdag Web platform aims to provide structural design capabilities through a browser-based interface. However, at the beginning of the internship, support for axially loaded column design was not available in the web version of Osdag. This limited users to desktop-only workflows and restricted accessibility.

The objective of this task was to develop an axially loaded column design module for Osdag Web, enabling users to perform column design calculations through a web-based interface while maintaining consistency with the Osdag Desktop design logic.

4.2 Task 2: Tasks Done

The following tasks were carried out as part of this activity:

- Designed the backend logic for axially loaded column design for Osdag Web.
- Ensured alignment of design checks with IS 800:2007 provisions.
- Structured input and output data flow for web-based execution.

- Integrated the column design logic with the Osdag Web framework.
- Tested the module using multiple input cases to validate correctness.

4.2.1 Refactoring of Core Design Logic

The core design logic for axially loaded columns was adapted from the desktop version to suit the stateless execution model of the web platform. Functions were reorganized to accept structured input data and return results in a format suitable for API-based communication.

4.2.2 Web-Based Architecture

The column design module was implemented as a backend service that processes user inputs received from the web interface and returns design results in a structured format. This separation of frontend and backend logic ensures scalability and ease of maintenance.

4.2.3 API Integration and Data Flow

User inputs such as section properties, axial load, and effective length parameters are passed from the web interface to the backend through API calls. The backend processes these inputs, performs design checks, and returns results including axial capacity and utilization ratios.

4.2.4 User Interface Design

A simple and intuitive user interface was designed for the Osdag Web platform to allow users to input column design parameters. The interface displays design results clearly, highlighting governing checks and design status.

4.2.5 Resource and UI Updates

Necessary updates were made to web resources to include the axially loaded column module in the Osdag Web navigation. Labels, input fields, and result displays were added to ensure a smooth user experience.

4.2.6 Testing and Verification

The module was tested using various combinations of section properties and axial loads. The web-based results were cross-verified with Osdag Desktop outputs to ensure consistency and correctness.

4.3 Task 2: Documentation

4.3.1 Directory Structure

The Osdag Web column design module follows a modular directory structure separating frontend and backend components.

4.3.2 Module Workflow Summary

- User inputs column design parameters through the web interface.
- Inputs are sent to the backend using API calls.
- Design calculations are performed for axially loaded columns.
- Results are returned and displayed on the web interface.

Chapter 5

Internship Task 3: Refactoring and UI Integration of Column-to-Column and Beam-to-Beam Connection Modules

5.1 Task 3: Problem Statement

The Osdag Desktop application contains multiple moment and splice connection modules such as beam-to-beam end plate, column end plate, and column cover plate connections. Initially, these modules suffered from inconsistent GUI routing, duplicated input-loading logic, and improper tab handling, which affected usability and maintainability.

The objective of this task was to refactor the connection modules related to beam-to-beam and column-to-column connections and integrate them cleanly into the Osdag Desktop user interface with consistent navigation, tab management, and restoration of previously saved design inputs.

5.2 Task 3: Tasks Done

The following tasks were carried out as part of this activity:

- Refactored GUI handler functions for beam-to-beam and column-to-column con-

nection modules.

- Integrated Beam–Beam End Plate, Column End Plate, and Column Cover Plate modules into the main window.
- Standardized loading of previous design inputs using OSI files.
- Ensured consistent tab creation, naming, and navigation behavior.
- Improved CAD hover label assignment for connection components.

5.2.1 Refactoring of Beam-to-Beam Connection Modules

The Beam–Beam End Plate and Beam Cover Plate moment connection modules were refactored to follow a unified GUI loading pattern. Each module is instantiated using a `CustomWindow` wrapper and added dynamically to the tab widget with an appropriate title. This ensured consistent behavior across all beam-to-beam connections.

5.2.2 Refactoring of Column-to-Column Connection Modules

Column End Plate and Column Cover Plate (bolted and welded) connection modules were integrated using a common workflow. The refactoring ensured that design inputs saved in previous sessions were automatically restored, improving continuity and user experience.

5.2.3 GUI Integration and Module Navigation

The main window was updated to route user actions based on selected card titles. Each connection module is loaded through a dedicated handler function, which clears the existing layout, initializes the backend module, restores previous inputs, and attaches the module to the active tab.

5.2.4 Python Code

A simplified version of the GUI integration logic for column and beam connection modules is shown below:

Listing 5.1: GUI Integration for Beam and Column Connection Modules

```
from osdag_core.design_type.connection.beam_beam_end_plate_splice
    import BeamBeamEndPlateSplice
from osdag_core.design_type.connection.column_end_plate import
   ColumnEndPlate
from osdag_core.design_type.connection.column_cover_plate import
   ColumnCoverPlate
from osdag_core.design_type.connection.column_cover_plate_weld
   import ColumnCoverPlateWeld

def open_column_end_plate_connection(self):
    title = "Column End Plate"
    self.clear_layout(self.main_widget_layout)
    CC_end_plate = CustomWindow(title, ColumnEndPlate, parent=
       self)

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

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

    self.main_widget_layout.addWidget(CC_end_plate)
```

5.2.5 CAD Logic and Hover Interaction Fixes

Hover dictionaries were correctly linked to the CAD widget to enable component-level inspection of beams, plates, and bolts. The refactoring ensured that hover labels are

assigned consistently regardless of the active connection module.

5.2.6 Testing and Validation

All beam-to-beam and column-to-column connection modules were tested using multiple input cases. GUI navigation, tab behavior, CAD visualization, and OSI file restoration were verified to ensure stable and consistent behavior across modules.

5.3 Task 3: Documentation

5.3.1 Directory Structure

The refactored modules follow the standard Osdag directory structure:

```
src/
|-- osdag_core/
|   |-- design_type/
|       |-- connection/
|           |-- beam_beam_end_plate_splice.py
|           |-- column_end_plate.py
|           |-- column_cover_plate.py
|           |-- column_cover_plate_weld.py
|-- osdag_gui/
|   |-- main_window.py
```

5.3.2 Module Workflow Summary

- User selects a connection module from the Osdag home screen.
- The corresponding GUI handler loads the module in a new tab.
- Previous design inputs are restored using OSI files.
- CAD models are generated with hover-enabled interaction.
- Design results are displayed and can be saved or exported.

Chapter 6

Internship Task 4: Refactoring and UI Support for Simply Supported and Cantilever Beam Modules

6.1 Task 4: Problem Statement

The Osdag Desktop application provides flexural member design capabilities for simply supported and cantilever beams. However, the existing implementation had inconsistent GUI routing, repeated logic for restoring previous design inputs, and limited integration with the main tab-based workflow of the application.

The objective of this task was to refactor the flexural member modules for simply supported and cantilever beams and integrate them cleanly into the Osdag Desktop user interface with consistent navigation, tab handling, and restoration of last design inputs.

6.2 Task 4: Tasks Done

The following tasks were performed as part of this activity:

- Integrated Simply Supported and Cantilever Beam design modules into the main GUI.
- Refactored module loading logic to follow a unified workflow.
- Enabled automatic restoration of previous design inputs using OSI files.

- Ensured consistent tab creation and naming for flexural member modules.
- Tested GUI interaction and module behavior for both beam types.

6.2.1 Core Design Logic Cleanup

The flexural member backend logic was reused without modifying the underlying design calculations. The refactoring focused on improving how the modules are instantiated and connected to the GUI, ensuring separation of design logic from interface handling.

6.2.2 Boundary Condition Handling

Separate backend classes were used to represent different boundary conditions. The `Flexure` class handles simply supported beams, while the `Flexure_Cantilever` class manages cantilever beam behavior. This separation ensures clarity in design workflows and correct interpretation of support conditions.

6.2.3 GUI Integration

Dedicated handler functions were implemented to load each flexural member module. When a user selects a beam type, the existing layout is cleared, the corresponding module is initialized using a `CustomWindow`, and the module is added to the active tab with an appropriate title.

6.2.4 Python Code

A simplified version of the GUI integration logic for flexural member modules is shown below:

Listing 6.1: GUI Integration for Flexural Member Modules

```
from osdag_core.design_type.flexural_member.flexure import
    Flexure
from osdag_core.design_type.flexural_member.flexure_cantilever
    import Flexure_Cantilever


def open_flexure_member(self):
```

```
    title = "Simply Supported Beam"
    self.clear_layout(self.main_widget_layout)
    flexure_ss = CustomWindow(title, Flexure, parent=self)

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

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

    self.main_widget_layout.addWidget(flexure_ss)

def open_flexure_cantilever_member(self):
    title = "Cantilever Beam"
    self.clear_layout(self.main_widget_layout)
    flexure_c = CustomWindow(title, Flexure_Cantilever, parent=
       self)

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

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

    self.main_widget_layout.addWidget(flexure_c)
```

6.2.5 Testing and Validation

Both simply supported and cantilever beam modules were tested using multiple input cases. GUI navigation, tab switching, input restoration, and result generation were verified to ensure stable and consistent behavior across different boundary conditions.

6.3 Task 4: Documentation

6.3.1 Directory Structure

The flexural member modules follow the standard Osdag directory structure:

```
src/
|-- osdag_core/
|   |-- design_type/
|       |-- flexural_member/
|           |-- flexure.py
|           |-- flexure_cantilever.py
|-- osdag_gui/
|   |-- main_window.py
```

6.3.2 Module Workflow Summary

- User selects a flexural member type from the Osdag interface.
- The corresponding module is loaded into a new tab.
- Previous design inputs are restored using OSI files.
- Design calculations are performed based on boundary conditions.
- Results are displayed and can be saved for future use.

Chapter 7

Internship Task 5: Testing, Bug Fixing, and Validation of Connection Modules

7.1 Task 5: Problem Statement

During the development and refactoring of multiple connection and flexural member modules in Osdag Desktop, several functional and UI-related issues were identified. These issues affected button actions, design generation, CAD hover interaction, and overall user workflow. In addition, inconsistencies were observed in restoring previous design inputs and handling preference-based UI elements.

The objective of this task was to systematically test various design modules, identify and fix reported issues, validate design generation workflows, and ensure stable behavior across connection modules.

7.2 Task 5: Tasks Done

The following activities were performed as part of testing and validation:

- Tested button functionality across multiple connection modules.
- Verified design generation and output consistency for different input cases.
- Identified and fixed issues related to UI preferences and input handling.

- Improved CAD hover labels for beams, columns, plates, bolts, welds, and stiffeners.
- Resolved multiple GitHub issues and contributed fixes through pull requests.

7.3 Task 5: Module Testing and Validation

Extensive testing was carried out for beam-to-beam and column-to-column connection modules, including end plate, cover plate (bolted and welded), and flexural member designs. Each module was tested for correct input processing, successful design execution, and accurate result display.

Special attention was given to validating the behavior of UI buttons, ensuring that design calculation, CAD visualization, and report generation actions executed without errors.

7.4 Task 5: Bug Fixes and Improvements

Several critical issues were identified and resolved during testing. Key fixes included:

- Corrected function signatures to ensure compatibility with updated UI callbacks.
- Fixed preference handling logic for flange plate and end plate configurations.
- Enhanced hover dictionary population for CAD components to display detailed design information.
- Resolved issues related to restoring previous design inputs using OSI files.
- Improved stability of design image rendering and output button behavior.

7.5 Task 5: Code Refactoring Highlights

The fixes were implemented across multiple modules, including beam–beam end plate splice, beam cover plate (bolted and welded), column end plate, and column cover plate modules. Hover dictionaries were populated with detailed information for beams, columns, plates, bolts, welds, and stiffeners to improve CAD interaction and clarity.

7.6 Task 5: Testing of Design Generation

Design generation was tested for various combinations of section properties, loads, and connection preferences. Generated designs were verified for numerical correctness, CAD visualization accuracy, and consistency with IS 800 design provisions. Both newly refactored modules and previously existing modules were validated to ensure no regression issues were introduced.

7.7 Task 5: Version Control and Issue Resolution

All fixes and improvements were committed to the Osdag GitHub repository. Multiple reported issues were resolved and merged through a pull request after successful review.

- Issues Resolved: #61, #58, #21
- Pull Request: #89
- Files Modified: 7
- Lines Changed: 235 additions, 18 deletions
- Status: Successfully merged into the test branch

7.8 Task 5: Documentation

Relevant documentation updates were made to reflect bug fixes, improved UI behavior, and enhanced CAD hover interaction. These updates help users better understand design outputs and interaction with connection components.

7.9 Task 5: Summary

This task ensured the robustness and reliability of the Osdag Desktop application by thoroughly testing design workflows, fixing critical issues, and validating design generation across multiple modules. The successful resolution and merging of reported issues significantly improved user experience and software stability.

Chapter 8

Conclusions

8.1 Tasks Accomplished

During the course of the semester-long internship, a wide range of structural design, software development, and validation activities were successfully carried out for both the Osdag Desktop and Osdag Web platforms. The work undertaken during the internship was focused on improving the robustness, usability, and extensibility of the Osdag software by refactoring existing modules, adding new design capabilities, enhancing user interface behavior, and performing extensive testing and validation.

One of the major technical contributions of the internship was the refactoring of the compression member module for axially loaded columns in Osdag Desktop. The existing logic was reorganized to improve readability, validation flow, and maintainability, while ensuring compliance with the provisions of IS 800:2007. The refactored module was thoroughly tested using multiple input combinations to verify correctness and stability. In parallel, an axially loaded column module was developed and integrated for the Osdag Web platform, enabling consistent design workflows across desktop and web environments.

Significant effort was also devoted to the refactoring and integration of structural connection modules. Column-to-column and beam-to-beam connection modules, including end plate and cover plate connections, were refactored to improve calculation flow, UI responsiveness, and CAD interaction. These modules were successfully integrated into the Osdag Desktop graphical interface with proper tab handling, input persistence, and output visualization. Special attention was given to ensuring correct CAD rendering

and hover-based information display for individual components such as beams, columns, plates, bolts, welds, and stiffeners.

Support for flexural members, including simply supported beams and cantilever beams, was enhanced through consistent UI integration and improved handling of design inputs, outputs, and validation messages. The design workflows for these modules were tested under various loading and section configurations to ensure reliable performance and accurate results.

A substantial portion of the internship involved systematic testing and verification of multiple modules across the Osdag Desktop application. This included testing of tension welded connections, fin plate connections, base plate connections, and flexural member modules using both manual input and OSI-based input restoration. Numerous functional, logical, and user interface–related issues were identified during testing, including problems related to button behavior, design generation, conditional input visibility, CAD hover interaction, and preference handling.

All identified issues were carefully analyzed, fixed, and validated. The resolved issues and improvements were submitted to the Osdag open-source repository through multiple GitHub pull requests. These contributions were reviewed, merged, and tracked through resolved GitHub issues, resulting in measurable improvements in software stability, usability, and maintainability. Overall, the tasks accomplished during the internship contributed directly to strengthening the quality and reliability of the Osdag software used by practicing engineers and students.

8.2 Skills Developed

The internship provided extensive exposure to both structural engineering principles and real-world software development practices. On the technical front, a deeper understanding of steel structure design as per IS 800:2007 was developed through hands-on implementation and testing of compression members, flexural members, and various structural connection modules.

Strong proficiency was gained in Python programming and object-oriented design through working on a large and evolving codebase. The internship involved refactoring legacy code, improving modularity, and implementing cleaner and more maintainable

design patterns. This experience significantly enhanced the ability to understand, modify, and extend complex engineering software.

Practical experience was also gained in graphical user interface development using PyQt, including layout management, event handling, and user interaction workflows. In addition, exposure to CAD integration using OpenCascade helped in understanding 3D geometry handling, model visualization, and component-level hover interaction, which are critical for engineering design software.

Beyond technical skills, the internship played an important role in developing professional and collaborative skills. These included systematic debugging of complex systems, writing clear and well-documented code, maintaining code quality through testing, and using version control systems such as Git effectively. Regular interaction through GitHub issues and pull requests improved understanding of collaborative software development, code review processes, and issue tracking in open-source projects.

Overall, the internship was a comprehensive and enriching learning experience that significantly strengthened both structural engineering knowledge and software development proficiency. The skills and experience gained during this internship provide a strong foundation for future work in engineering software development and structural design.

Chapter A

Appendix

A.1 Internship Work Reports

This section contains the detailed day-wise work reports maintained during the internship period. The reports include daily tasks performed, hours worked, testing activities, refactoring efforts, and documentation progress. These records provide a chronological overview of the work carried out throughout the internship.

Internship Work Report

Name: Yugh Juneja
Project: Osdag
Internship: FOSSEE Semester Long Internship (Autumn 2025)

Date	Day	Task	Hours
01-Oct-2025	Wednesday	Set up Osdag Desktop development environment and verified execution of existing design modules.	4
02-Oct-2025	Thursday	Installed Osdag dependencies and validated basic workflows for structural design modules.	4
03-Oct-2025	Friday	Explored Osdag codebase structure and reviewed column and connection design modules.	5
04-Oct-2025	Saturday	Fixed environment-related issues and validated Python package compatibility.	4
06-Oct-2025	Monday	Studied compression member design logic for axially loaded columns as per IS 800:2007.	4
07-Oct-2025	Tuesday	Reviewed existing Column Design module and identified refactoring requirements.	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.	0
13-Oct-2025	Monday	Exam Break.	0
14-Oct-2025	Tuesday	Began refactoring compression member module for axially loaded columns.	6
15-Oct-2025	Wednesday	Improved validation logic and axial capacity checks in column design module.	5
16-Oct-2025	Thursday	Integrated refactored column module with Osdag Desktop GUI.	6
17-Oct-2025	Friday	Tested column design module using multiple input combinations.	5
18-Oct-2025	Saturday	Discussed refactored column design workflow with mentor and incorporated feedback.	4
20-Oct-2025	Monday	Designed backend logic for axially loaded column module for Osdag Web.	6
21-Oct-2025	Tuesday	Implemented web-compatible input and output handling for column design.	5
22-Oct-2025	Wednesday	Tested Osdag Web column module and compared results with desktop version.	5
23-Oct-2025	Thursday	Improved error handling and result formatting for web-based column design.	5
24-Oct-2025	Friday	Validated Osdag Web column design outputs and documented workflow.	7
25-Oct-2025	Saturday	Submitted updates and discussed Osdag Web column module with mentor.	5
27-Oct-2025	Monday	Started refactoring beam-to-beam connection modules for UI integration.	6

28-Oct-2025	Tuesday	Refactored Beam–Beam End Plate connection logic and GUI routing.	7
29-Oct-2025	Wednesday	Integrated CAD hover interaction for beam connection components.	6
30-Oct-2025	Thursday	Began refactoring column-to-column connection modules.	7
31-Oct-2025	Friday	Completed refactoring of column end plate connection module.	6
01-Nov-2025	Saturday	Tested column end plate module and restored OSI-based input loading.	5
03-Nov-2025	Monday	Refactored column cover plate bolted connection module.	6
04-Nov-2025	Tuesday	Refactored column cover plate welded connection module.	5
05-Nov-2025	Wednesday	Verified CAD visualization and hover labels for column connections.	6
06-Nov-2025	Thursday	Refactored butt joint bolted connection module.	7
07-Nov-2025	Friday	Tested butt joint bolted connection with multiple design cases.	6
08-Nov-2025	Saturday	Verified OSI file loading and persistence across connection modules.	5
10-Nov-2025	Monday	Refactored butt joint welded connection module.	7
11-Nov-2025	Tuesday	Validated welded connection calculations and outputs.	8
12-Nov-2025	Wednesday	Refactored flexural member module for simply supported beams.	8
13-Nov-2025	Thursday	Improved flexural member code structure and design checks.	6
14-Nov-2025	Friday	Tested simply supported beam module and verified results.	7
15-Nov-2025	Saturday	Added UI support for cantilever beam module.	6
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.	6
21-Nov-2025	Friday	Improved calculation flow and GUI consistency for base plate module.	7
22-Nov-2025	Saturday	Updated UI behavior and CAD visualization for base plate connections.	6
24-Nov-2025	Monday	Debugged edge cases in base plate design calculations.	5
25-Nov-2025	Tuesday	Completed refactoring of base plate connection module.	6
26-Nov-2025	Wednesday	Performed regression testing of base plate module.	7
27-Nov-2025	Thursday	Documented base plate refactoring and testing results.	6
28-Nov-2025	Friday	Submitted pull request for base plate module improvements.	7

29-Nov-2025	Saturday	Verified mentor feedback and updated base plate module accordingly.	6
01-Dec-2025	Monday	Tested end plate and header plate connection modules with random inputs.	7
02-Dec-2025	Tuesday	Verified design generation using OSI input restoration.	6
03-Dec-2025	Wednesday	Tested beam-to-beam cover plate welded connection module.	7
04-Dec-2025	Thursday	Validated welded connection results against expected outputs.	6
05-Dec-2025	Friday	Verified design reports and CAD visualization accuracy.	5
06-Dec-2025	Saturday	Reported and fixed issues related to tension welded module.	6
08-Dec-2025	Monday	Documented testing observations and inconsistencies across modules.	7
09-Dec-2025	Tuesday	Raised GitHub issues and tracked fixes through pull requests.	6
10-Dec-2025	Wednesday	Assisted in debugging and resolving reported issues.	7
11-Dec-2025	Thursday	Retested modules after fixes and validated stability.	8
12-Dec-2025	Friday	Prepared detailed testing summary for mentor review.	8
13-Dec-2025	Saturday	Reported testing feedback and verified UI behavior.	6
15-Dec-2025	Monday	Organized code changes and documentation for final report.	6
16-Dec-2025	Tuesday	Drafted final internship report sections.	7
17-Dec-2025	Wednesday	Consolidated internship work logs and technical notes.	6
18-Dec-2025	Thursday	Continued writing and formatting internship report.	6
19-Dec-2025	Friday	Refined report content and corrected technical details.	7
20-Dec-2025	Saturday	Incorporated mentor feedback into report draft.	6
22-Dec-2025	Monday	Final proofreading and technical verification of report.	6
23-Dec-2025	Tuesday	Prepared final version of internship report.	7
24-Dec-2025	Wednesday	Compiled appendix and contribution summary.	5
25-Dec-2025	Thursday	Assisted junior contributors with Osdag setup and guidance.	6
26-Dec-2025	Friday	Investigated remaining open issues and documented findings.	7
27-Dec-2025	Saturday	Continued analysis of unresolved issues.	5
29-Dec-2025	Monday	Added final edits and completed report submission material.	6
30-Dec-2025	Tuesday	Submitted report draft to mentor for final review.	5
31-Dec-2025	Wednesday	Final internship report submission.	6

A.2 GitHub Contributions and Pull Requests

During the internship, multiple issues were identified, fixed, and contributed to the Osdag open-source repository through GitHub pull requests. These contributions involved UI improvements, refactoring of structural design modules, CAD hover handling fixes, testing, and validation.

A.2.1 List of Pull Requests Submitted

The following pull requests were submitted and reviewed as part of the internship:

- **#89** — Resolved Issues #61, #58, and #21 related to input dock callbacks, conditional visibility, and hover handling.
- **#71** — Added UI support for Column-to-Column moment connection modules.
- **#17** — Fixed column CAD integration, output dock behavior, and capacity-related UI updates.
- **#6** — Fixed integration issues in axially loaded column module.

A.2.2 Summary of Contributions

These pull requests resulted in improvements across multiple Osdag modules, including beam-to-beam connections, column-to-column connections, flexural members, and testing workflows. The contributions enhanced usability, stability, and maintainability of the Osdag Desktop application and were successfully reviewed and merged into the repository.

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.