FOSSEE Winter Internship Report

On

Refactoring of Member Properties and Development of CAD Visualization System for OsdagBridge

Submitted by

Sreejesh S

2nd Year BE Student, Department of CSE With CyberSecurity

Sri Eshwar College of Engineering

Coimbatore

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Nidhi Khare

Aditya Donde

Parth Karia

February 27, 2026

Acknowledgments

I sincerely thank everyone who supported and guided me throughout my FOSSEE Winter Internship, which made this project a valuable and enriching learning experience. I am deeply grateful to my mentors, Nidhi Khare and Aditya Donde, for their continuous guidance, technical insights, and constant encouragement during the OsdagBridge refactoring and CAD visualization work. I also thank the Osdag project team, especially Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia, for fostering a supportive and collaborative work environment.

I respectfully acknowledge Prof. Siddhartha Ghosh, Principal Investigator of Osdag, Department of Civil Engineering, IIT Bombay, for providing me with the opportunity to contribute to open-source engineering development. I also express my sincere gratitude to Prof. Kannan M. Moudgalya, Principal Investigator of the FOSSEE project, Department of Chemical Engineering, IIT Bombay, for his leadership and vision in facilitating this internship program. I extend my appreciation to the FOSSEE Managers, Usha Viswanathan and Vineeta Parmar, and their team for their efficient coordination and consistent support.

I gratefully acknowledge the support of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education, Government of India. I also thank my fellow interns and colleagues for their cooperation and constructive discussions throughout the internship. Finally, I express my sincere gratitude to the authorities of Sri Eshwar College of Engineering, including my Department Head and Principal, for their encouragement and academic support.

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.

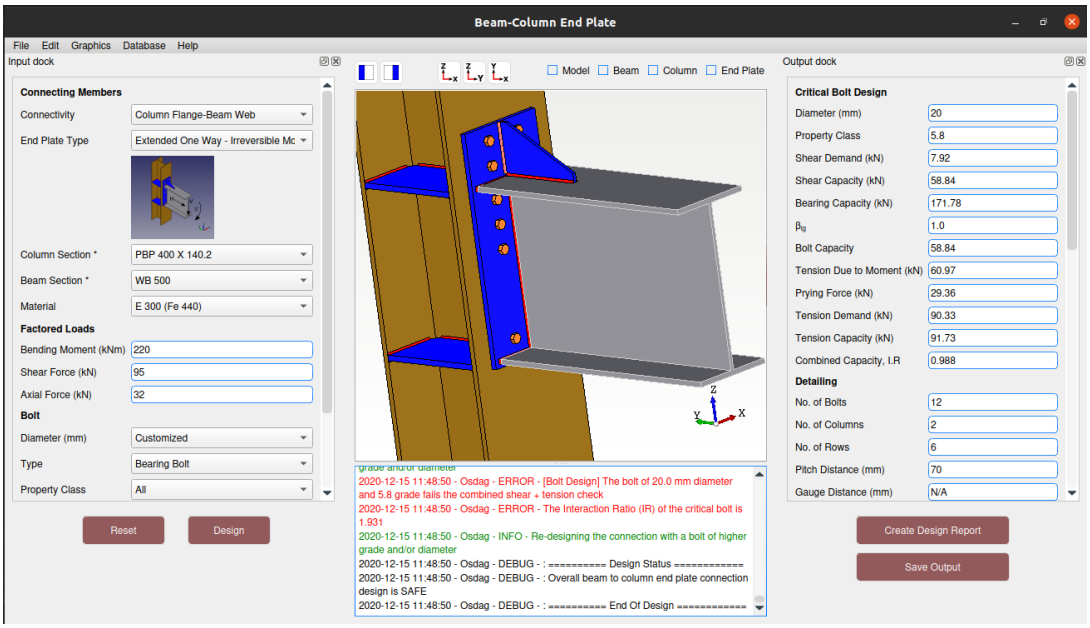


Figure 1.2: Osdag GUI

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 to implement the Group Design module as a standalone web application with a React frontend and a Django REST backend. The scope covered a full Basic Inputs workflow, a spreadsheet-style popup for custom loading parameters, accurate validation rules for geometry, materials, and loads, and an API layer that supports location-based lookup and server-side validation. The UI had to follow the Osdag-web color theme, use reusable form components, and keep a static bridge cross-section visualization on the right panel.

2.2 Tasks Done

I implemented the end-to-end screening task based on the provided requirements, focusing on both the frontend behavior and backend endpoints. The work included:

- Built the two-panel layout with left-side tabs (Basic Inputs, Additional Inputs placeholder) and a non-interactive right-side bridge cross-section image.
- Implemented the Type of Structure dropdown with the "Other structures not included" message and hard-disable of all remaining inputs.
- Created two mutually exclusive Project Location modes: (A) Location Name mode with State and District dropdowns plus auto-filled wind, seismic, and temperature

values; (B) Custom Loading mode with a spreadsheet-style popup for wind, seismic, and temperature inputs.

- Ensured all auto-filled values are displayed in green and that mode toggles correctly disable the other mode.
- Implemented validation rules for span, carriageway width, footpath, and skew angle warning as specified by IRC 24 (2010).
- Added material selection inputs for girder steel, cross bracing steel, and deck concrete.
- Implemented the Modify Additional Geometry popup with synchronized fields (girder spacing, number of girders, and deck overhang) and correct auto-adjustment logic based on overall width.
- Added backend API endpoints for location lookup, custom loading, materials, and geometry validation with normalized responses.

2.3 Implementation Highlights

2.3.1 Frontend

The React UI uses reusable components (Dropdown, Input, FormSection, PopupModal) to keep behavior consistent across tabs. I implemented state-driven UI for the two location modes, summary fields, and the geometry popup. Validation errors are displayed inline, and warnings (such as skew angle beyond $\pm 15^{\circ}$) are shown without blocking input.

2.3.2 Backend

The Django REST backend exposes endpoints for location lookups, custom loading inputs, materials list, and geometry validation. The geometry validation endpoint returns both error states and auto-adjusted values for the popup so the UI remains consistent. A CSV/JSON data source is supported for location-based inputs, with a fallback to hardcoded values when the dataset is unavailable.

2.4 Validation and Rules Implemented

- Span outside 20–45 m returns "Outside the software range."
- Carriageway width must be ≥ 4.25 m and < 24 m.
- Skew angle outside $\pm 15^\circ$ shows a warning requiring detailed analysis per IRC 24 (2010).
- Overall width = carriageway width +5 m.
- Overhang and spacing are each $<$ overall width.
- (Overall Width − Overhang)/Spacing = No. of Girders with auto-adjustment on any field change.

2.5 Outcome

The screening task resulted in a fully working standalone module with a clean, Osdag-themed UI, complete validations, and a functional API layer. The spreadsheet popup and auto-adjustment logic were fully integrated, and the Basic Inputs flow supports both location-based and custom loading data with consistent feedback to the user.

2.6 My Contributions (Repository Work Summary)

In addition to implementing the functional requirements, I made incremental improvements through multiple commits focused on UI consistency, validations, state management, and backend support.

2.6.1 Osdag-Screening-Task (main branch)

The complete source code and commit history for this task can be accessed at: github. Based on the commit history, my contributions in this repository include:

- Implemented core bridge visualization components (2D cross-section view) and integrated the view into the application layout.

- Added bridge view modes along with validation indicators/highlights and improved schematic dimensions for better clarity.
- Refactored UI layout and form controls for improved responsiveness and consistent branding.
- Integrated enhanced dropdown behavior (including disabling search where needed) and improved accessibility/styling.
- Implemented geometry validation and adjustment utilities and enhanced the geometry popup with equation display.
- Added backend APIs to support location lookup and improved reference data handling.
- Added report-generation capability using jsPDF and improved error handling around the bridge view.
- Updated database usage by migrating from SQLite to PostgreSQL and updating backend requirements (including psycopg).
- Improved summary card styling and dropdown menu behavior for better overall UX.

Chapter 3

Refactoring and Enhancement of Member Properties

3.1 Task 1: Problem Statement

The existing Member Properties and Visualization systems in OsdagBridge faced several limitations that hindered the structural design workflow:

- **Manual Data Entry**: Member IDs were manually entered, leading to naming inconsistencies and potential errors across design tabs.
- **Lack of Visual Feedback**: The absence of a real-time, engineering-grade preview meant users could not visually verify section properties or weld configurations before analysis.
- **State Persistence Issues**: UI inputs were lost when switching between different girder pairs, requiring redundant and error-prone data re-entry.
- **Inconsistent Annotations**: Standard UI labels lacked the professional clarity required for structural drawings, such as proper subscripts (t_{fw}) and units.

3.2 Task 1: Tasks Done

To address these issues, a comprehensive refactoring of the Member Properties and a ground-up implementation of a CAD Visualization engine were executed:

- **Quality Assurance & Bug Identification**: Conducted extensive testing on the existing Fin Plate connection module. By identifying edge-case failures and UI inconsistencies, I contributed to the overall stability of the Osdag software suite, ensuring that the new bridge components integrate seamlessly with existing connection logic.

- **CAD-Style Visualization Engine**: Developed the `RolledSectionPreview` widget, a professional-grade 2D rendering system for both rolled (catalog) and welded (custom) sections. This system provides real-time feedback for engineering parameters.

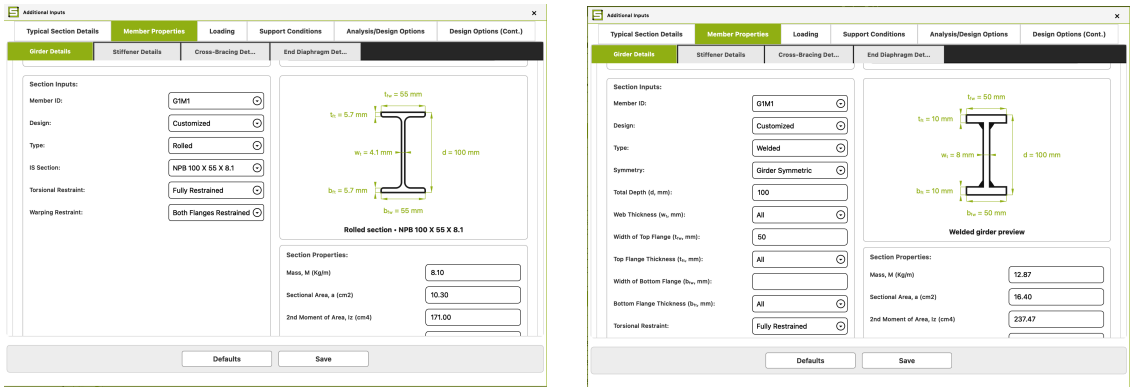


Figure 3.1: Professional 2D rendering for Rolled (Left) and Welded (Right) sections.

- **Dynamic Member ID Generation**: Developed a software-driven calculation engine to generate canonical Member IDs (e.g., $B\{pair\}M\{member\}$) based on the total span and spacing, ensuring naming consistency across the project.

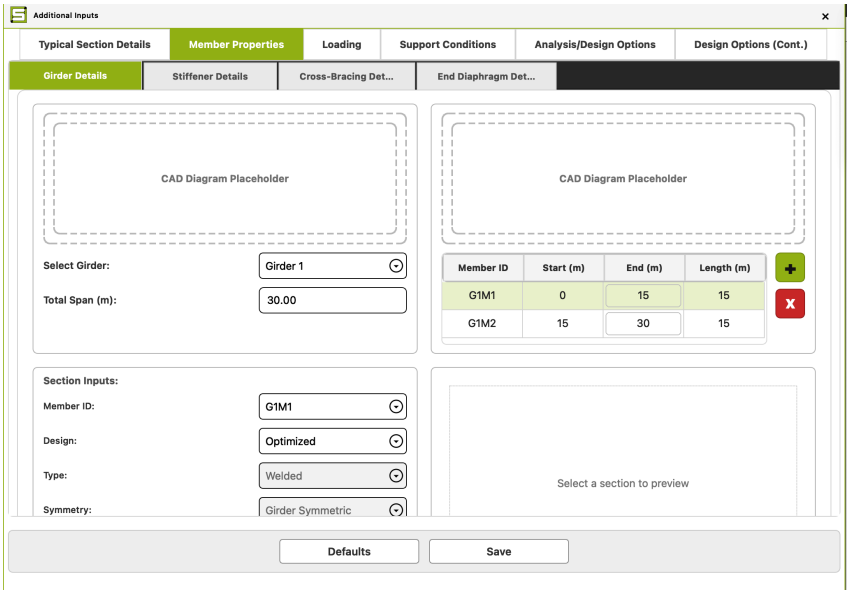


Figure 3.2: Automatic generation of Member IDs based on Span and Spacing logic.

- **Per-Member State Persistence**: Designed a robust architecture using Python dictionaries to store and restore UI configurations for every unique member key, preventing data loss when navigating between girders.
- **Bulk Actions**: Introduced an "Apply to All" functionality in the Stiffener Details tab to allow rapid configuration of multiple members while respecting "Optimized" design constraints.

3.3 Task 1: Python Code & Logic

This section details the implementation of the CAD geometry engine and the dynamic member calculation logic.

3.3.1 Description of the Script

The implementation focuses on two core engineering functionalities:

- **Vector-Based CAD Geometry**: Uses unit and normal vectors to render direction-aware, filled triangular arrowheads with a professional 1:3 ratio.
- **Parametric Rendering**: Employs `QPainterPath` to draw complex I-beam geometries including root fillets, toe radii, and quadratic Bézier curves for weld profiles.

3.3.2 Python Code: CAD Geometry & Member Logic

Listing 3.1: CAD Visualization and Member Logic in OsdagBridge

```
import math
from PySide6.QtCore import QPointF, Qt
from PySide6.QtGui import QPainter, QColor, QPen

# --- CAD Dimension Palette Implementation ---
DIMENSION_PALETTE = {
    "tfw": QColor("#1e88ff"), "tft": QColor("#00a152"),
    "bfw": QColor("#ff8f00"), "bft": QColor("#f4511e"),
    "d":   QColor("#5e35b1"), "wt":  QColor("#00838f")
}

```

```
12 # --- Professional CAD Arrow Geometry ---
13 def _draw_arrow_head(self, painter: QPainter, tip: QPointF, direction:
       QPointF, color: QColor):
14     length = math.hypot(direction.x(), direction.y())
15     if length == 0: return
16     unit = QPointF(direction.x() / length, direction.y() / length)
17     normal = QPointF(-unit.y(), unit.x())
18
19     # Professional 1:3 ratio geometry (Length=1.3, Width=0.9)
20     base = tip + unit * (self._arrow_size * 1.3)
21     left = base + normal * (self._arrow_size * 0.45)
22     right = base - normal * (self._arrow_size * 0.45)
23
24     painter.save()
25     painter.setBrush(color)
26     painter.setPen(Qt.NoPen)
27     painter.drawPolygon([tip, left, right])
28     painter.restore()
29
30 # --- Dynamic Member Calculation ---
31 def _cross_bracing_member_count(self) -> int:
32     span_m = self._get_total_span_m()
33     spacing_m = self._get_cross_bracing_spacing_m()
34     if not span_m or not spacing_m: return 1
35
36     # Formula: m = (Span / Spacing) - 1
37     raw = (span_m / spacing_m) - 1.0
38     return max(1, int(math.floor(raw + 1e-9)))
```

3.3.3 Explanation of the Code

- **Line 6-10**: Defines a standard engineering palette. This reduces cognitive load by assigning specific colors to specific geometric properties (e.g., Green always represents Top Flange Thickness).
- **Line 13-28**: Implements vector-based rendering. By calculating the `normal` vector, the system ensures that arrowheads are perfectly perpendicular to the dimension lines regardless of the section's aspect ratio or zoom level.

- **Line 31-37**: The geometric count logic utilizes a $1e-9$ epsilon margin to handle floating-point precision issues, ensuring the correct number of bracing members is generated when the spacing is an exact divisor of the span.

3.4 Task 1: Documentation

The following directory structure highlights the modular separation between the core logic, UI tabs, and the newly developed CAD utility.

3.4.1 Directory Structure

```
osdagbridge
├── core
│   └── bridge_types
│       └── plate_girder
│           └── ui_fields.py (Dynamic Field Logic)
└── desktop
    └── ui
        ├── dialogs
        │   └── tabs
        │       └── sub_tabs
        │           └── section_properties
        │               ├── cross_bracing_details_tab.py
        │               └── stiffener_details_tab.py
        └── utils
            └── rolled_section_preview.py (CAD Engine)
```

Implementation Note: The CAD engine (`rolled_section_preview.py`) provides a parametric geometry system that supports both standard `Intg_osdag.sqlite` catalog sections and custom welded sections. It features adaptive scaling to ensure that members ranging from 100mm to 2000mm in depth are rendered with consistent visual clarity and margin spacing.

Chapter 4

Generalization of Input/Output Docks and State Management

4.1 Task 2: Problem Statement

The initial architecture of OsdagBridge suffered from significant scalability and usability issues regarding data handling:

- **Hardcoded UI Logic**: The `InputDock` contained over 500 lines of repetitive code where every widget was manually instantiated and configured. Adding a single new input parameter required modifications across multiple files.
- **Statelessness**: The application lacked a unified state store. Data entered in dialogs (like Additional Inputs) or specific tabs was often lost when the dialog was closed or when switching between views.
- **Tight Coupling**: Frontend widgets were tightly coupled with backend variables, making it difficult to implement features like "Save Project" or "Undo/Redo".
- **Maintenance Overhead**: There was no central definition for default values or validation rules, leading to inconsistencies between the UI defaults and the solver's expectations.

4.2 Task 2: Tasks Done

To resolve these architectural debts, a complete refactoring of the Input/Output system was executed, moving to a dynamic, data-driven approach:

- **Backend State Architecture**: Implemented a centralized `FrontendData` class in `ui_fields.py`. This acts as a single source of truth, storing all input/output values in a dictionary-based structure accessible by any component.
- **Dynamic Widget Factory**: Refactored `InputDock` and `OutputDock` to generate UI components dynamically based on metadata definitions (Label, Type, Default Value). This reduced the codebase size by approximately 400 lines while improving reliability.
- **Reactive Two-Way Binding**: Established a signal-slot architecture where UI changes automatically update the backend state, and backend updates dynamically refresh the UI.
- **Hierarchical Persistence**: Implemented a standardized `collect_data()` and `restore_data()` interface across all complex design tabs (Girder, Stiffener, Cross Bracing). This allows the application to capture nested state trees (e.g., specific settings for Member M1 vs. Member M2) for saving and loading.
- **Smart Defaults & Placeholders**: Developed logic to auto-populate fields from metadata defaults and introduced `PlaceholderSectionPreview` widgets to improve the user experience during empty states.

4.3 Task 2: Python Code

This section presents the core implementation of the centralized state store and the dynamic widget generation logic.

4.3.1 Description of the Script

The script illustrates two key architectural patterns:

- **The State Store**: A dictionary-backed system that handles data retrieval with safe defaults.
- **The Factory Pattern**: A method that inspects field metadata (type, validation rules) and instantiates the correct Qt widget automatically.

4.3.2 Python Code

Listing 4.1: Centralized State Management and Dynamic Widget Generation

```
# --- Centralized State Store (ui_fields.py) ---
class FrontendData:
    def __init__(self):
        self._input_state: dict[str, object] = {}

    def set_input_value(self, key: str, value):
        """Update state and notify observers."""
        self._input_state[key] = value
        # Signals would trigger here to update dependent UI parts

    def get_input_value(self, key: str, default=None):
        """Retrieve state with safe fallback."""
        return self._input_state.get(key, default)

    def prime_defaults(self, definitions):
        """Auto-populate state from field metadata."""
        for key, _, _, metadata in definitions:
            if key not in self._input_state and metadata:
                default = metadata.get("default")
                if default is not None:
                    self._input_state[key] = default

# --- Dynamic Widget Factory (input_dock.py) ---
def _create_input_widget(self, key, field_type, values, metadata):
    """Factory method to generate widgets from metadata."""
    widget = None

    if field_type == "ComboBox":
        widget = NoScrollComboBox()
```

```
30          if values: widget.addItems(values)
31          # Restore saved state
32          saved_val = self.backend.get_input_value(key)
33          if saved_val: widget.setCurrentText(str(saved_val))
34          # Bind UI to Backend
35          widget.currentTextChanged.connect(
36              lambda text: self.backend.set_input_value(key, text)
37          )
38
39      elif field_type == "LineEdit":
40          widget = QLineEdit()
41          saved_val = self.backend.get_input_value(key)
42          if saved_val: widget.setText(str(saved_val))
43          # Bind UI to Backend
44          widget.editingFinished.connect(
45              lambda: self.backend.set_input_value(key, widget.text())
46          )
47
48      return widget
```

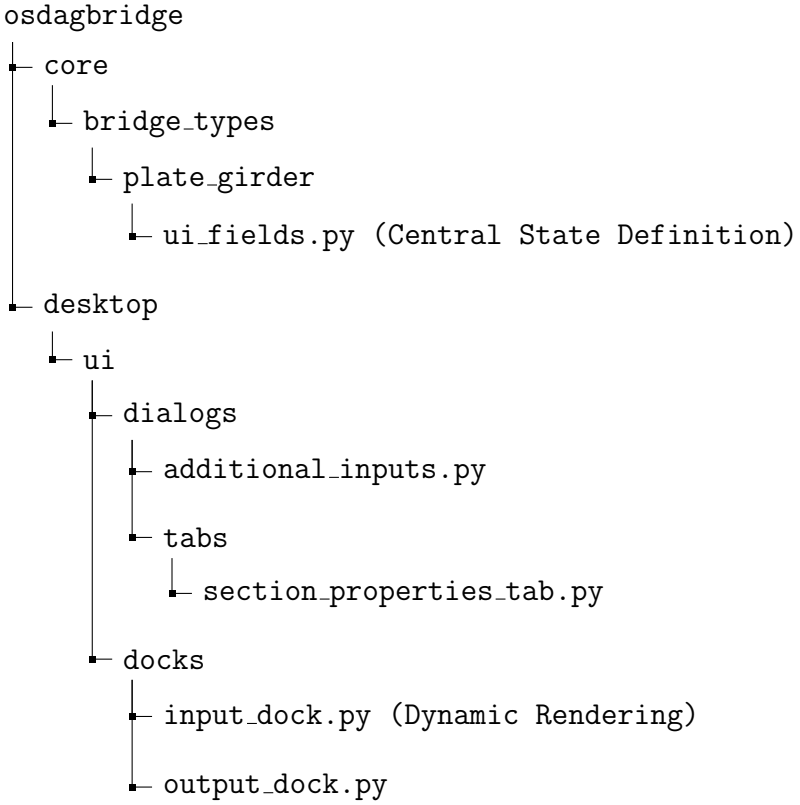
4.3.3 Explanation of the Code

- **Line 15-21**: The `prime_defaults` method ensures that even before a user interacts with the UI, the backend state is populated with sensible default values defined in the metadata. This eliminates "NoneType" errors in the solver.
- **Line 24-48**: The `_create_input_widget` function acts as a factory. Instead of manually writing 50+ blocks of code for 50 inputs, this single function loops through a configuration list. It handles initialization, state restoration (pulling from backend), and event binding (pushing to backend) in one place.

4.4 Task 2: Documentation

The following directory structure highlights the files refactored to support the new state management architecture.

4.4.1 Directory Structure



Implementation Note: The `ui_fields.py` file now serves as the centralized "schema" for the application. Any new input field added to this file is automatically rendered in the Input Dock, validated, and persisted to the state file without requiring any changes to the UI code.

Chapter 5

Conclusions

5.1 Tasks Accomplished

Over the course of this fellowship, I addressed critical architectural and usability challenges within OsdagBridge, transforming the interface into a dynamic, state-aware engineering tool. Key tasks included:

- **Member Properties Refactoring**: Re-engineered the logic for Cross Bracing, End Diaphragms, and Stiffeners. I implemented a software-driven calculation engine for Member IDs and a persistence architecture to prevent data loss.
- **CAD Visualization Engine**: Built the `RolledSectionPreview` widget from scratch. This uses vector mathematics to render 2D sections with ISO-standard dimensions and adaptive scaling for visual design verification.
- **I/O Architecture Generalization**: Refactored the `InputDock` and `OutputDock` using a metadata-driven approach. This eliminated 400+ lines of boilerplate code and established a reactive two-way binding system.
- **Quality Assurance**: Conducted stress-testing and bug-tracking for the **Fin Plate Connection Module**. By identifying edge-case failures in geometry and solver outputs, I improved the software's reliability.

5.2 Skills Developed

This internship provided a rigorous environment for applying software engineering to structural design. Key skills developed include:

- **Advanced GUI Development**: Deepened expertise in PySide6, specifically in custom widget creation, signal-slot mechanisms, and rendering performance with `QPainter`.
- **Architecture & Design Patterns**: Applied the **Observer**, **Factory**, and **Singleton** patterns to solve scalability and state management issues.
- **Computational Geometry**: Mastered vector math and coordinate transformations to ensure dimension lines and arrowheads render accurately across orientations.
- **Testing and Debugging**: Sharpened systematic debugging skills through mass testing of the Fin Plate module, ensuring software implementations aligned with IS 800:2007 standards.
- **Engineering Domain Knowledge**: Gained insights into plate girder design and the importance of precise technical dimensioning in structural drawings.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report			
Name:	Sreejesh S		
Project:	Osdag		
Internship:	FOSSEE Winter Fellowship 2025		
DATE	**DAY**	**TASK**	**Hours Worked**
01-Dec-2025	Monday	Onboarding Meet	2
02-Dec-2025	Tuesday	Overview on Version Control for Osdag	3
03-Dec-2025	Wednesday	Linux setup for installation	3
04-Dec-2025	Thursday	Osdag web setup	4
05-Dec-2025	Friday	Osdag desktop setup	4
06-Dec-2025	Saturday	Meeting regarding how to test the new version	3
07-Dec-2025	Sunday	Practiced Working of Osdag	2
08-Dec-2025	Monday	Discussed about project allocation	3
09-Dec-2025	Tuesday	Worked on Mass Testing	6
10-Dec-2025	Wednesday	Worked on Fin plate connection module	8
11-Dec-2025	Thursday	Testing on Tension member welded to end gusset	8
12-Dec-2025	Friday	Gone through the GitHub issues	5
13-Dec-2025	Saturday	Gone through the bridge module repo	6
14-Dec-2025	Sunday	Meeting regarding the task assignment	3
15-Dec-2025	Monday	Worked on Developing seed data for girder	6
16-Dec-2025	Tuesday	Implemented girder properties loader and UI preview system	8
17-Dec-2025	Wednesday	Dynamic Diagram ui enhancements	8
18-Dec-2025	Thursday	CAD-style dimension annotations with color coding + Meeting	8
19-Dec-2025	Friday	Cad diagram Enhancement and refactoring	8
20-Dec-2025	Saturday	Add label details to input as subscript	7
21-Dec-2025	Sunday	Gone through the osdag repo for reference	5
22-Dec-2025	Monday	Member Properties overview and refactoring girder details	8
23-Dec-2025	Tuesday	Viewed osdag codebase for reference and learnt the structure	9
24-Dec-2025	Wednesday	Viewed the dynamic iterative code in osdag	9
25-Dec-2025	Thursday	Viewed the bridge module codebase to implement dynamic rendering	10
26-Dec-2025	Friday	Redesigned the welded part	12
27-Dec-2025	Saturday	Redesigned the rolled type with curves and meeting	10
28-Dec-2025	Sunday	Improvised the design of rolled section	8
29-Dec-2025	Monday	Added weld visibility toggle and dimension rendering	6
30-Dec-2025	Tuesday	Code review and bug fixes	4
31-Dec-2025	Wednesday	SVG resources and InputDock dynamic field generation	8
01-Jan-2026	Thursday	Holiday - Partial work on documentation	2

Internship Work Report			
Name:	Sreejesh S		
Project:	Osdag		
Internship:	FOSSEE Winter Fellowship 2025		
DATE	**DAY**	**TASK**	**Hours Worked**
02-Jan-2026	Friday	Holiday - Partial work on code review	2
03-Jan-2026	Saturday	Holiday - Partial work on planning	2
04-Jan-2026	Sunday	Holiday - Partial work on testing	2
05-Jan-2026	Monday	Holiday - Partial work	2
06-Jan-2026	Tuesday	OutputDock refactoring and section rendering improvements	8
07-Jan-2026	Wednesday	Code optimization and testing	6
08-Jan-2026	Thursday	Implement 1/3 ratio arrow geometry enhancement	8
09-Jan-2026	Friday	Testing and validation of CAD preview system	7
10-Jan-2026	Saturday	Refactored girder properties completed	8
11-Jan-2026	Sunday	Refactored section inputs of member properties	7
12-Jan-2026	Monday	WIP: Additional inputs dialog enhancements	8
13-Jan-2026	Tuesday	Continued work on additional inputs	8
14-Jan-2026	Wednesday	Refactored girder detail's welded and rolled type	10
15-Jan-2026	Thursday	Testing and bug fixes for girder details	8
16-Jan-2026	Friday	Code review and optimization	7
17-Jan-2026	Saturday	Major refactor: Input/Output dock dynamic rendering	12
18-Jan-2026	Sunday	Testing dynamic dock rendering	6
19-Jan-2026	Monday	Bug fixes and UI improvements	8
20-Jan-2026	Tuesday	Documentation and code cleanup	7
21-Jan-2026	Wednesday	Testing and validation	8
22-Jan-2026	Thursday	UI polish and refinements	8
23-Jan-2026	Friday	Complete refactor of girder details tab	12
24-Jan-2026	Saturday	Testing girder details refactor	8
25-Jan-2026	Sunday	Bug fixes and improvements	6
26-Jan-2026	Monday	State restoration and member ID handling enhancements	10
27-Jan-2026	Tuesday	Testing state management system	8
28-Jan-2026	Wednesday	UI improvements and validation	8
29-Jan-2026	Thursday	Backend state management implementation + End Diaphragm UI	12
30-Jan-2026	Friday	Testing and integration	8
31-Jan-2026	Saturday	Final testing and documentation	7

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.