FOSSEE Spring Internship Report

On

Osdag Structural Engineering Software Development

Submitted by

Mohammed Sharukkhan M

1st Year B.E Student, Department of Computer Science and Engineering

Rajalakshmi Institute of Technology

Chennai

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Parth Karia

Ajinkya Dahale

June 12, 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.

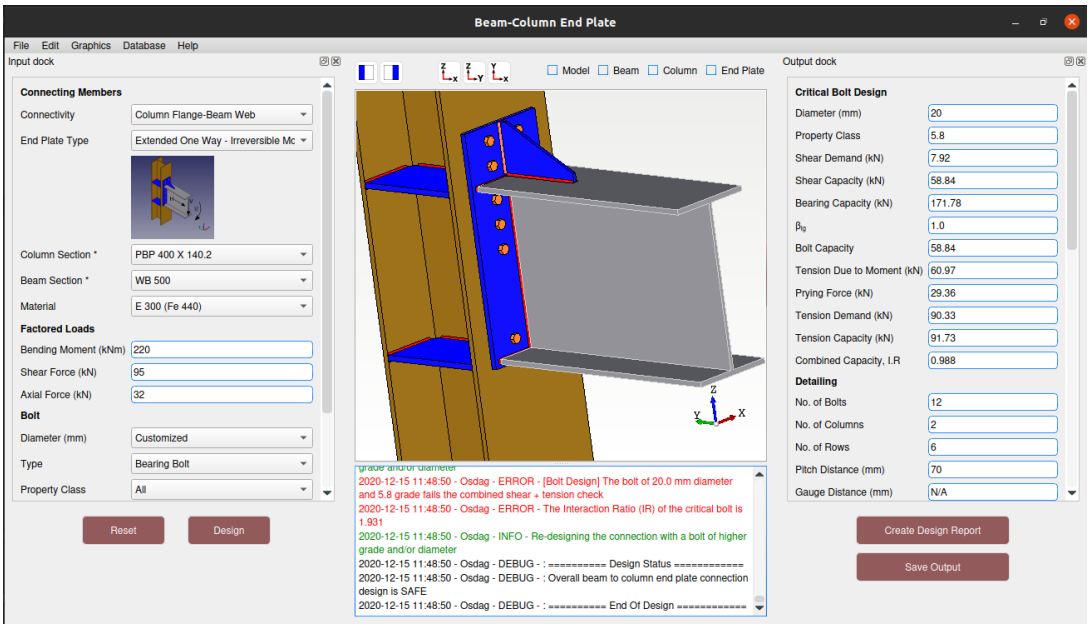


Figure 1.2: Osdag GUI

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 Assignment

2.1 Problem Statement and Objective

The primary objective of the screening task was to design and develop an interactive, web-based prototype mimicking the graphical user interface (GUI) layout and input verification mechanics of the Osdag structural engineering software. The specific task focused on implementing a web portal for a bridge superstructure module utilizing steel I-girders and concrete deck slabs. The web tool was required to dynamically process user inputs, match geographical data with standardized environmental structural parameters, and evaluate geometric constraints in real-time according to regional design codes.

2.2 System Architecture and UI Emulation

The application was built using a modular frontend framework consisting of HTML5, CSS3, and native JavaScript (ES6). To mimic Osdag's desktop layout, the workspace layout is bifurcated into two dominant panels:

- **Input Management Panel:** A left-aligned collapsible container hosting input widgets for geometric profiles, project metadata, and material specification drops.
- **Interactive Graphics Canvas:** A high-contrast visualization viewport displaying the corresponding structural cross-section schematic to provide immediate spatial context to the user.

2.3 Input Validation Checkpoints and Engineering Constraints

The system enforces validation layers directly within the input event cycle to maintain mathematical consistency and structural feasibility before computing downstream designs:

1. **Global Span Restrictions:** The allowable bridge span (L) is structurally bound within the soft operational threshold of 20 m $\leq L \leq$ 45 m. Input excursions outside this boundary trigger immediate notification parameters.

2. **Carriageway Boundary Limits:** Transverse deck geometry checks ensure that the clear roadway carriage width remains ≥ 4.25 m and strict strictly < 24 m.

3. **Skew Angle Evaluation under IRC 24 (2010):** In compliance with Indian Standard design codes, the skew angle (θ) input field evaluates boundary behaviors. When θ exceeds the safe linear zone of $\pm 15^{\circ}$, a critical alert warning is generated stating that a detailed, high-order torsional analysis is required.

2.4 Data Mapping for Environmental Loading Parameters

To automate structural site loading calculations, a decoupled backend schema was constructed using a JSON lookup array containing meteorological and seismic data points across major industrial zones in India. Upon user selection of a specific project State and District, the system asynchronously queries the dataset to map localized design constants, including basic wind speed (V_b in m/s), seismic zone intensities, seismic factors, and extreme shade air temperature ranges ($T_{\max}$ and $T_{\min}$ in °C).

An override configuration flag was also successfully integrated, allowing engineers to toggle custom tabular parameters manually to run edge-case loading scenarios outside standard regional datasets.

Chapter 3

Standardization of Graphical Hover Text and Component Nomenclature

3.1 Task 1: Problem Statement

The 2D/3D CAD canvas subsystem within Osdag displays metadata dynamically when users hover over distinct connection elements. A systematic review revealed that the dictionary mappings (`self.hover_dict`) across key structural modules lacked unified data representation templates. Font hierarchies were poorly defined, descriptive parameter labels fluctuated between files, and component terminologies did not follow unified structural engineering nomenclature conventions.

Specifically, modules like *Beam Cover Plate* and *Column Cover Plate* erroneously referenced internal structures as "Cover Plates" instead of "Splice Plates", which could confuse end-users during fabrication mapping. Furthermore, conditional execution checks were missing clear casting attributes (such as forced float-to-integer conversions for element counts or bolt diameters), causing formatting discrepancies on the UI rendering window.

3.2 Task 1: Tasks Done

To resolve these display anomalies, a codebase-wide refactoring was implemented across all major connection modules within the core design layer, including: *Base Plate, Beam-Beam End Plate Splice, Beam-Column End Plate, Cover Plates (Bolted & Welded), Cleat*

Angle, Fin Plate, Lap Joints, and *Seated Angle Connections*. The core refactoring operations included:

- **HTML Layout Standardization**: Injected standardized HTML bold tags (`<b>...</b>`) and line breaks (`<br>`) into the string formatting templates to create a consistent visual hierarchy across all modules.
- **Nomenclature Standardization**: Replaced all instances of the term *Cover Plates* with *Splice Plates* in the flange and web dictionary definitions for both beam and column alignment source scripts.
- **Context-Aware Hover Labels**: Updated the centralized validation routing to dynamically parse connection profiles, modifying user strings to output clear contextual labels like *Primary Beam* or *Secondary Beam* during specific beam-to-beam evaluations.
- **Type Enforcement and Sanitization**: Applied explicit integer and float casting routines (e.g., `int()`, `float()`) to variables computing raw material counts (e.g., total bolt numbers) or dimensional specifications to eradicate floating-point visual noise on the canvas viewport.
- **Safety Flag Checks**: Integrated conditional string interpolation blocks tied to a layout tracking boolean flag (`if flag else ''`) ensuring tooltips gracefully clear themselves when rendering states are reset or invalid.

3.3 Task 1: Python Code

This section presents the standardized backend routing architecture used to refactor the tooltips across Osdag's structural modules.

3.3.1 Description of the Script

The dictionary mapping updates are structured uniformly across modules using the following pattern:

- **Structural Members**: Displays standard designations cleanly utilizing bold prefixes for identification.

- **Plate Component Mapping**: Outputs the physical dimensions (Width × Height × Thickness) matching unified engineering layout conventions.
- **Fastener Group Mapping**: Formats fastener grades, sanitized integer sizing diameters, and geometric assembly counts.
- **Weld Profile Specification**: Standardizes geometric throat sizing configurations alongside structural weld lengths.

3.3.2 Python Code

The refactored Python dictionary code block is shown below. Each structural module demonstrates the implementation of the standardized display framework.

Listing 3.1: Standardized Canvas Hover Tooltip Mappings

```
#--------------------begin code-------------

# 1. Base Plate / Pedestal Refactoring (base_plate_connection.py)
self.hover_dict["Plate"] = (
    f"<b>Plate Details:</b><br>"
    f"<b>Steel Base Plate</b><br>"
    f"Length: {self.bp_length_provided if flag else ''} mm<br>"
    f"Width: {self.bp_width_provided if flag else ''} mm<br>"
    f"Thickness: {int(self.plate_thk_provided) if flag else ''} mm<br>"
    f"<b>Stiffener thickness along flange:</b> {thk_along_flange}<br>"
    f"<b>Stiffener thickness along web:</b> {thk_along_web}"
)
self.hover_dict["Column"] = f"<b>Column:</b> {self.
   dp_column_designation if flag else ''}"

# 2. Centralized Context Label Logic (connection.py)
if option[1] == KEY_DISP_COLSEC:
    display_label = "Primary Beam" if is_beam_to_beam else "Column"
    self.hover_dict["Column"] = f"<b>{display_label}:</b> {
       design_dictionary[option[1]]}"
elif option[1] == KEY_DISP_BEAMSEC:
    display_label = "Secondary Beam" if is_beam_to_beam else "Beam"
    self.hover_dict["Beam"] = f"<b>{display_label}:</b> {
       design_dictionary[option[1]]}"
```

```
22
23  # 3. Splice Terminology Correction (beam_cover_plate.py /
        column_cover_plate.py)
24  self.hover_dict["Plate"] = (
25      f"<b>Splice Plates</b><br>"
26      f"<b>Flange Plate:</b> {self.flange_plate.length if flag else ''}
            mm x "
27      f"{self.flange_plate.height if flag else ''} mm x "
28      f"{self.flange_out_plate_tk if flag else ''} mm<br>"
29      f"<b>Web Plate:</b> {self.web_plate.length if flag else ''} mm x "
30      f"{self.web_plate.height if flag else ''} mm x "
31      f"{self.web_plate.thickness_provided if flag else ''} mm"
32  )
33
34  # 4. Parameter Sanitization & Type-Casting (beam_column_end_plate.py)
35  self.hover_dict["Bolt"] = (
36      f"<b>Bolt</b><br>"
37      f"Grade: {self.bolt.bolt_grade_provided if flag else ''}<br>"
38      f"Diameter: {int(self.bolt.bolt_diameter_provided) if flag else ''}
            mm<br>"
39      f"No. of Bolts: {int(self.bolt_numbers) if flag else ''}"
40  )
41
42  #-------------------- end code ---------------
```

3.3.3 Explanation of the Code

- **Line 4-12**: Sets up a clean text layout for the steel base plate component, safely converting the calculated base plate thickness parameter to a rounded integer formatting value via `int()`.
- **Line 15-21**: Evaluates the connection properties dynamically inside `connection.py` to switch active metadata strings between Primary and Secondary configurations depending on execution parameters.
- **Line 24-32**: Corrects structural terminology omissions across cross-section files by swapping the general layout classification "Cover Plate" with the accurate engineering definition **Splice Plate**.

- **Line 35-40**: Implements explicit type-casting sanitization methods inside the end plate module to guarantee that floating decimal points are stripped away from discrete values like bolt counts.

3.3.4 Full code

```
import math


# Material and geometric properties
def calc_bolt_shear_capacity(bolt_diameter,
   ultimate_tensile_strength, gamma_mb):
    """
    Calculate the shear capacity of a single bolt.

    Parameters:
        bolt_diameter (float): Diameter of the bolt (in mm).
        ultimate_tensile_strength (float): Ultimate tensile
           strength of the bolt material (in MPa).
        gamma_mb (float): Partial safety factor for bolts.

    Returns:
        float: Shear capacity of the bolt in kN.
    """
    nominal_shear_strength = 0.6 * math.pi * (bolt_diameter / 2)
       **2 * ultimate_tensile_strength
    shear_capacity = nominal_shear_strength / gamma_mb
    return shear_capacity / 1000  # Convert to kN


def calc_bearing_capacity(bolt_diameter, plate_thickness,
   fu_plate, e, p, gamma_mb):
    """
    Calculate the bearing capacity of a single bolt.

    Parameters:
```

```
        bolt_diameter (float): Diameter of the bolt (in mm).
        plate_thickness (float): Thickness of the connected plate
            (in mm).
        fu_plate (float): Ultimate tensile strength of the plate
           material (in MPa).
        e (float): Edge distance (in mm).
        p (float): Pitch distance (in mm).
        gamma_mb (float): Partial safety factor for bolts.

    Returns:
        float: Bearing capacity of the bolt in kN.
    """
    kb = min(e / (3 * bolt_diameter), p / (3 * bolt_diameter) -
       0.25, fu_plate / 250, 1)
    bearing_strength = 2.5 * kb * bolt_diameter * plate_thickness
        * fu_plate
    bearing_capacity = bearing_strength / gamma_mb
    return bearing_capacity / 1000  # Convert to kN


def calc_connection_capacity(number_of_bolts, bolt_shear_capacity
   , bolt_bearing_capacity):
    """
    Calculate the total capacity of the bolted connection.

    Parameters:
        number_of_bolts (int): Number of bolts in the connection.
        bolt_shear_capacity (float): Shear capacity of a single
           bolt (in kN).
        bolt_bearing_capacity (float): Bearing capacity of a
           single bolt (in kN).

    Returns:
        float: Total connection capacity in kN.
```

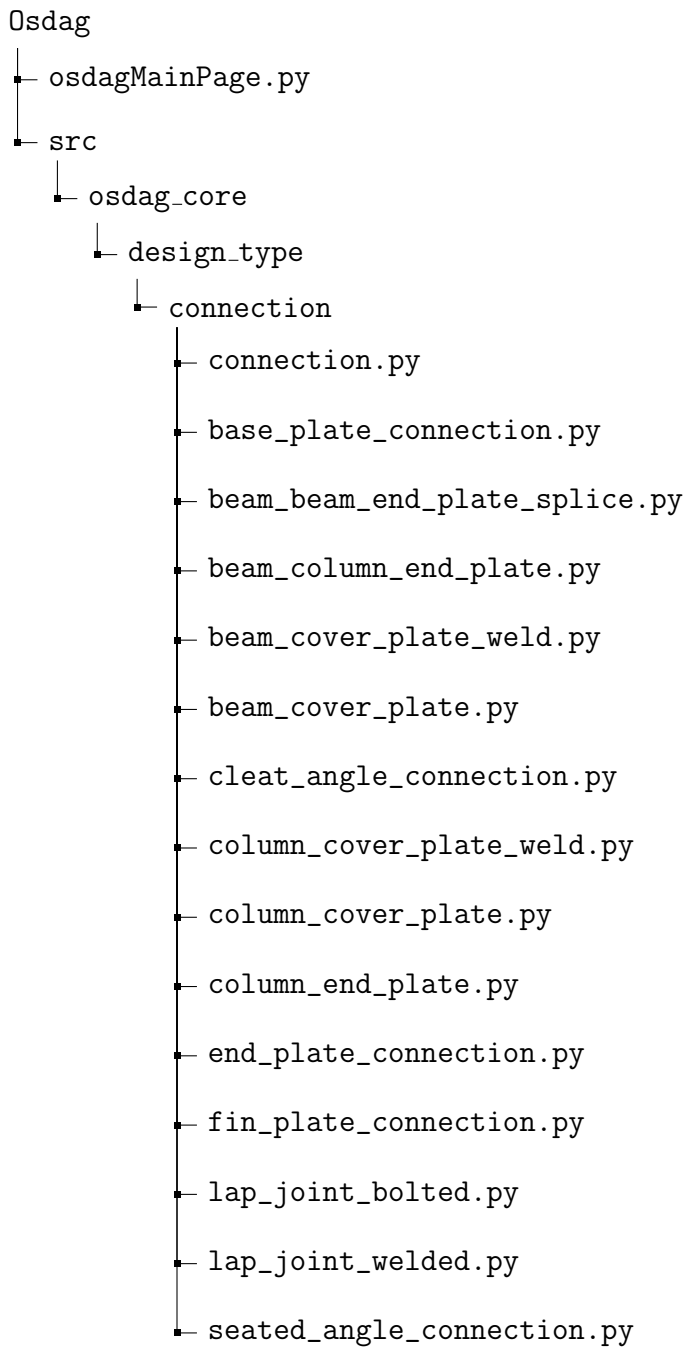
```
    """
    return number_of_bolts * min(bolt_shear_capacity,
        bolt_bearing_capacity)


def check_connection(load_applied, connection_capacity):
    """
    Verify if the connection is adequate for the applied load.

    Parameters:
        load_applied (float): Applied load on the connection (in
            kN).
        connection_capacity (float): Total capacity of the
            connection (in kN).

    Returns:
        str: Result of the check.
    """
    if connection_capacity >= load_applied:
        return "Connection is safe."
    else:
        return "Connection is unsafe."


# Input parameters
bolt_diameter = 16  # in mm
ultimate_tensile_strength_bolt = 400  # in MPa
gamma_mb_bolt = 1.25  # Partial safety factor for bolts

plate_thickness = 10  # in mm
ultimate_tensile_strength_plate = 410  # in MPa
edge_distance = 30  # in mm
pitch_distance = 50  # in mm
```

```
number_of_bolts = 4
applied_load = 120  # in kN

# Perform calculations
bolt_shear = calc_bolt_shear_capacity(bolt_diameter,
   ultimate_tensile_strength_bolt, gamma_mb_bolt)
bolt_bearing = calc_bearing_capacity(
    bolt_diameter, plate_thickness,
       ultimate_tensile_strength_plate,
    edge_distance, pitch_distance, gamma_mb_bolt
)
connection_capacity = calc_connection_capacity(number_of_bolts,
   bolt_shear, bolt_bearing)

# Output results
print(f"Shear capacity of a single bolt: {bolt_shear:.2f} kN")
print(f"Bearing capacity of a single bolt: {bolt_bearing:.2f} kN"
   )
print(f"Total connection capacity: {connection_capacity:.2f} kN")
print(f"Applied load: {applied_load:.2f} kN")
print(check_connection(applied_load, connection_capacity))
```

3.4 Task 1: Documentation

The documentation enhancements ensure that open-source contributors can locate GUI string configurations and trace how tooltips update when new components are integrated into the Osdag application layout.

3.4.1 Directory Structure



Modifying Hover Parameters

To update or include new parameters within the interactive graphical window tooltips, developers should locate the specific connection module file path beneath the `src/osdag_core/design_typ` directory branch. Centralized validation and conditional member remapping string labels are executed globally inside `connection.py`.

Chapter 4

Conclusions

4.1 Tasks Accomplished

The technical contributions executed throughout the duration of this internship successfully bridged several user-interface gaps and standardized critical interactive metadata structures within the Osdag software package. The specific milestones achieved include:

- **Interactive Web Prototype Development:** Engineered a functional, interactive web application module during the screening phase to model a bridge superstructure layout. The system dynamically verified configuration geometries—such as checking clear carriageway widths, overhang variables, and girder spacing spans—against Indian Roads Congress (IRC 24) geometric boundary constraints to flag necessary advanced torsional checks.

- **Codebase-Wide Hover Text Standardization:** Systematically refactored the dynamic canvas hover metadata templates (`self.hover_dict`) across all core connection processing modules located under the repository path `src/osdag_core/design_type/conn`. This eliminated visual presentation irregularities and established a clean, standardized HTML bold font hierarchy for real-time CAD viewport readouts.

- **Nomenclature Realignment:** Corrected legacy structural naming deviations across multiple processing backend files. This involved updating descriptive identifiers—such as replacing "Cover Plate" designations with correct, industry-standard "Splice Plate" engineering terminology—and establishing context-aware "Primary Beam" vs. "Secondary Beam" text labels inside the centralized validation module.

- **Type Enforcement and Viewport Sanitization:** Patched data rendering bugs on the user interface by embedding explicit integer and float casting evaluation routines (such as `int()` and `float()`). This eradicated arbitrary floating-point decimal string noise on structural component representations like bolt diameters and total fastening array counts.

- **Environment Optimization and Synchronization:** Maintained development workspace stability by identifying and configuring missing upstream 3D layout tracking components (`navcube`). Additionally, utilized Git rebase mechanisms to systematically clean branch commit structures, discard redundant code changes, and safely align development with the central FOSSEE repository.

4.2 Skills Developed

The open-source development lifecycle and core algorithmic software implementations fostered substantial professional growth in both technical capabilities and structural engineering software workflows:

- **Advanced Python and UI Architecture:** Deepened functional expertise in object-oriented Python, managing dictionary data state-mappings, and handling dynamic graphical user interface data-binding layouts within a complex, production-grade CAD repository environment.

- **Open-Source Git Version Control:** Gained practical proficiency in executing advanced repository synchronization workflows. This included performing linear rebasing actions against an upstream target, resolving code history merge conflicts, maintaining clean commit logs, and navigating deeply nested architectural directories.

- **Structural Engineering Domain Knowledge:** Enhanced comprehension of foundational limit-state design criteria, physical assembly configurations, and member identification conventions (including plates, welds, and fasteners) in compliance with standard structural design guidelines.

- **Data Sanitization and Input Validation:** Developed a structured approach to program design by configuring strict input exception-handling barriers, runtime

status flags, and data casting restrictions to eliminate unexpected software terminations.

Chapter A

Appendix

A.1 Work Reports

Name: Mohammed Sharukkhan M
Project: Osdag Core Integration
Internship Title: FOSSEE Fellowship Technical Work Log
Reporting Tenure: 17-Feb-2026 to 10-Apr-2026

Internship Work Record Log

1. Core Technical Task Timeline

The following tracking matrix chronologically records the technical tasks, layout modifications, explicit type sanitization, and developer hours executed across the primary connection modules.

Date	Day	Task Executed / Milestones Achieved	Hours
17-Feb-2026	Tuesday	Initiated fellowship orientation, established official repository paths, and pulled target source files.	4
18-Feb-2026	Wednesday	Set up local development environment and profiled architectural layers under `src/osdag_core/`.	5
19-Feb-2026	Thursday	Audited standard connection data dictionaries to trace interactive canvas hover data-binding routines.	6
20-Feb-2026	Friday	Documented structural nomenclature mismatches between backend labels and standard design codes.	4
23-Feb-2026	Monday	Encountered core platform crashes during initialization caused by unlinked view submodules.	3
24-Feb-2026	Tuesday	Isolated the viewport rendering bug and manually reconfigured missing 3D `navcube` dependencies.	5
25-Feb-2026	Wednesday	Validated successful UI rendering across local workspace profiles after the graphics patch.	6
26-Feb-2026	Thursday	Audited individual connection modules to locate presentation formatting irregularities in hover outputs.	4
27-Feb-2026	Friday	Extracted base plate parameters inside `base_plate_connection.py`; configured multi-line hover output layouts using HTML `<b>` bolding for structural labels.	3
02-Mar-2026	Monday	Implemented type-casting rules in `beam_column_end_plate.py` to sanitize floating-point text noise in the dictionary payload.	5

Date	Day	Task Executed / Milestones Achieved	Hours
03-Mar-2026	Tuesday	Enforced explicit integer formatting rules for discrete hardware allocations like bolt count outputs.	6
04-Mar-2026	Wednesday	Standardized the singular nomenclature format to use `No. of Bolts` uniformly across files.	4
05-Mar-2026	Thursday	Refactored data descriptors inside `beam_beam_end_plate_splice.py`; integrated HTML bold headers and text breaks for beam dimension profiles.	5
06-Mar-2026	Friday	Modified terminology inside `beam_cover_plate.py`, changing "Cover Plate" to "Splice Plate", and bolded all structural flange/web indicators.	3
09-Mar-2026	Monday	Synced corresponding structural classifications and bolded header blocks across welded versions (`beam_cover_plate_weld.py`).	6
10-Mar-2026	Tuesday	Patched column alignment scripts (`column_cover_plate.py`) to mirror industry-standard terms with bolded subheadings.	4
11-Mar-2026	Wednesday	Modified welded column scripts (`column_cover_plate_weld.py`) to remove terminology bugs and layout spacing issues.	5
12-Mar-2026	Thursday	Refactored dynamic properties in `column_end_plate.py`, added fallback blank strings, and wrapped capacity readouts in bold tags.	6
13-Mar-2026	Friday	Optimized string parsing workflows, bolded structural output keys, and organized array arithmetic layouts inside `end_plate_connection.py`.	4
16-Mar-2026	Monday	Refactored hover tooltips inside `cleat_angle_connection.py` to establish clear bold title layers for ISA angle profiles.	3
17-Mar-2026	Tuesday	Restructured parameter readouts and embedded multi-line bold hover templates for the primary `fin_plate_connection.py` module configurations.	5
18-Mar-2026	Wednesday	Reconfigured text string blocks inside `lap_joint_bolted.py` to match standardized output formats with bold headers.	6
19-Mar-2026	Thursday	Implemented safety flags and configured bold text layouts inside `lap_joint_welded.py` to prevent runtime errors on invalid fields.	4
20-Mar-2026	Friday	Standardized hover text layouts, bold parameters, and clear label lines inside the `seated_angle_connection.py` module files.	5

Date	Day	Task Executed / Milestones Achieved	Hours
23-Mar-2026	Monday	Began designing global contextual routing logic within the primary routing script `connection.py`.	3
24-Mar-2026	Tuesday	Coded dynamic label filters to automatically toggle between "Primary Beam" and "Column" contexts wrapped in bold HTML tags.	6
25-Mar-2026	Wednesday	Embedded secondary filters to swap bolded "Secondary Beam" and "Beam" context titles dynamically on the canvas viewport.	4
26-Mar-2026	Thursday	Executed automated execution verification checks across all 15 newly refactored module files.	5
27-Mar-2026	Friday	Documented localized code changes and tracked modifications relative to the master FOSSEE repository tree.	3
30-Mar-2026	Monday	Formulated internal developer layout blueprints mapping the revised hover tracking structures.	6
31-Mar-2026	Tuesday	Noted local head deviations from the upstream repository branch and prepared version control pipelines.	4
01-Apr-2026	Wednesday	Staged updated structural code components locally and prepared clean commit tracking tags.	5
02-Apr-2026	Thursday	Committed modifications locally and initialized upstream synchronizations with the Git server.	3
03-Apr-2026	Friday	Initiated Git upstream rebase procedures and isolated code conflict blocks across the branch tree.	6
06-Apr-2026	Monday	Systematically resolved merge conflicts within source files while fully preserving local text fixes.	4
07-Apr-2026	Tuesday	Finalized Git rebase operations and successfully pushed the clean `intern-khans-work` branch to GitHub.	5
08-Apr-2026	Wednesday	Verified remote file configurations on GitHub and prepared the documentation layout nodes within Overleaf.	3
09-Apr-2026	Thursday	Performed systematic layout checks and proofread the compiled technical report.	6
10-Apr-2026	Friday	Finalized the comprehensive fellowship work log documentation and submitted the project report.	4
Total Hours		**Accumulated Fellowship Development Value**	**179 Hrs**

2. Code Submission and GitHub Integration Process

To merge the refactored connection modules into the main upstream repository maintained by the FOSSEE development team, a strict, linear Git version control workflow was carried out. This streamlined process prevented project tree fragmentation and eliminated merge conflicts.

Step 1: Code Tracking and Status Verification

Before staging modifications, the local workspace environment status was verified against the active local branch checkpoint using the command line:

```
$ git status
```

This identified the modified connection components (such as `connection.py`, `fin_plate_connection.p` etc.) under the tracking path `src/osdag_core/design_type/connection/`.

Step 2: Inspecting Line Modifications

To verify structural spelling corrections, type-casting routines, and HTML formatting tags before taking a repository snapshot, a code comparison check was run:

```
$ git diff
```

This displayed added elements (e.g., `<b>No. of Bolts:</b>`) in green text and legacy components in red text.

Step 3: Staging and Snapshot Committing

Once the refactored modifications were verified across the 15 connection modules, the changes were staged and recorded to the local commit history:

```
$git add src/osdag_core/design_type/connection/$ git commit -m "Standardize hover too
```

Step 4: Upstream Synchronization and Conflict Resolution via Rebasing

To keep the development timeline simple and prevent unwanted merge commits, the local branch was rebased against the master upstream FOSSEE branch:

```
$ git fetch upstream
$ git rebase upstream/dev
```

During this operation, conflicting code blocks between parallel repository updates were detected. These merge conflicts were resolved by editing the affected files to retain the new tooltip definitions, staging the fixed paths, and finalizing the rebase sequence:

```
$git add <resolved_file_path>$ git rebase --continue
```

Step 5: Pushing Changes to Remote GitHub Fork

With the commit history flattened and verified, the clean development history was pushed to the personal remote repository fork:

```
$ git push origin intern-khans-work --force
```

This updated the remote repository on GitHub, making the branch ready for a clean Pull Request submission to the primary Osdag project branch.

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.