FOSSEE Winter Internship Report

On

Development and Enhancement of Block Shear and Tension Member Design Modules for Osdag

Submitted by

Ayushman Dwivedi

2nd Year B.Tech Student, Department of Computer Science and Engineering

Vellore Institute of Technology

Bhopal

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

May 17, 2026

Acknowledgments

I would like to express my heartfelt gratitude to everyone who supported and guided me throughout the successful completion of my project, Development and Enhancement of Block Shear & Tension Member Design Module for Osdag. First and foremost, I am deeply thankful to the project staff at the Osdag team — Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia — for their constant guidance, technical support, and encouragement at every stage of this work. I extend my sincere gratitude to the Osdag Principal Investigator, Prof. Siddhartha Ghosh, Department of Civil Engineering, IIT Bombay, whose vision and leadership made this project possible. I am also grateful to Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay, for his invaluable support and direction. I equally thank FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team for their coordination and administrative support throughout the project. I gratefully acknowledge the support of the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for facilitating this project and providing the necessary resources to carry it forward. I would also like to thank my colleagues who worked alongside me during this internship. Their collaboration, discussions, and shared enthusiasm made the journey both productive and enjoyable. I extend my sincere thanks to VIT Bhopal University, and particularly to Dr. Shriram R, Deputy Director — Placement and Training, for fostering an environment that encourages practical learning through such valuable internship opportunities. His efforts in bridging academia and industry have been truly instrumental in shaping this experience. Lastly, I am sincerely thankful to my department faculty and the entire academic community at VIT Bhopal University for their continuous support and encouragement throughout my academic journey.

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.

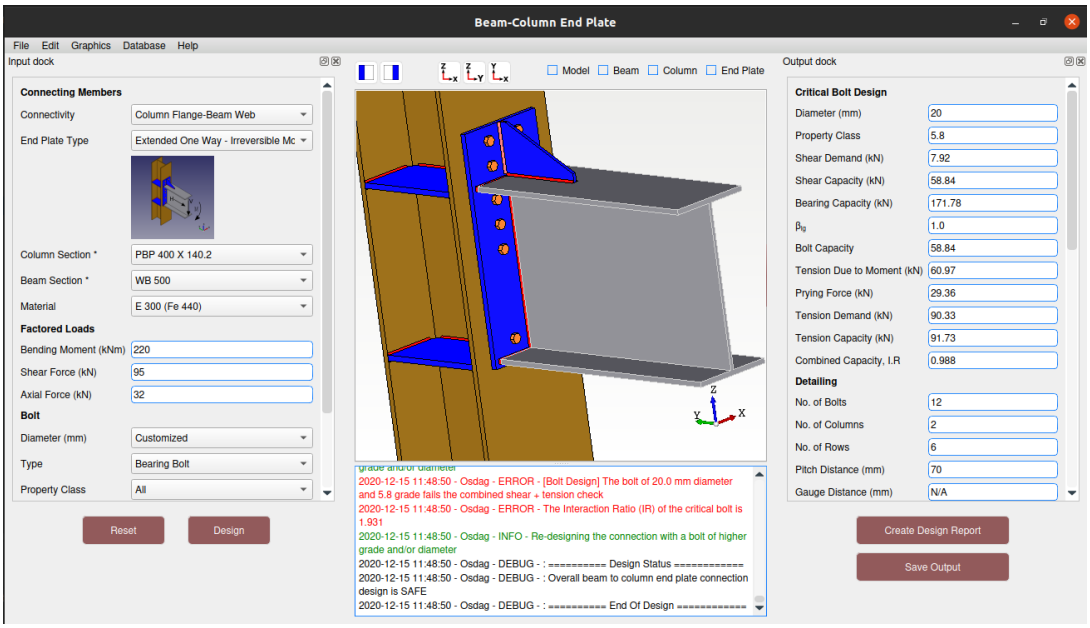

Figure 1.2: Osdag GUI

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 required the development of a parametric, modular 3D CAD model of a short-span steel girder bridge using the **pythonOCC (pythonocc-core)** library in Python. The model was to be assembled from individual component factories — including main longitudinal steel girders (I-sections), a concrete deck slab, and parapets — such that each part remains fully adjustable by simply changing named parameters at the top of the script.

The final deliverable was a runnable Python script, `bridge_model.py`, capable of building, visualizing, and optionally exporting the complete bridge geometry in STEP or BREP format.

2.2 Tasks Done

The following tasks were performed as part of the screening task:

1. **Study of Bridge Components** — Understood the structural layout of a short-span steel girder bridge, including the I-section main girders, concrete deck slab, and parapets, using the provided reference figures and problem description.

2. **Parameter Definition** — Defined all key geometric and layout parameters at the top of the script, including span length, number of girders, girder spacing, I-section

dimensions (depth, flange width, flange thickness, web thickness), deck thickness, deck overhang, and skew angle, using millimetres as the working unit.

3. **Component Factory Implementation** — Utilized and integrated the provided helper files:

   - `draw_i_section.py` — to generate 3D extruded I-section solids for the main girders.
   - `draw_rectangular_prism.py` — to generate the concrete deck slab as a rectangular prism.

4. **Assembly of Bridge Model** — Implemented the following assembly functions:

   - `build_girders()` — created and correctly positioned n girders along the bridge cross-section using parametric translations.
   - `build_deck()` — constructed the deck slab as a prism spanning the full bridge width.
   - `assemble_bridge()` — combined all components into a single OCC compound, placing each part accurately in 3D space using geometric transformations.

5. **Visualization** — Initialized the OCC display window, assigned appropriate colors to each component, set an isometric camera view, and applied zoom-to-fit for clear visual output.

6. **Export Functionality** — Implemented optional export of the full assembly to **STEP** and **BREP** file formats, controlled via boolean flags in the parameter section.

7. **Documentation and Submission** — Prepared the `bridge_model.py` script that runs without errors, along with a PDF report, video demonstration, GitHub repository, and a ZIP file containing all relevant project files.

Chapter 3

Internship Task 1: Enhancement of Fin Plate Shear Connection Module in Osdag

3.1 Task 1: Problem Statement

The first internship task focused on enhancing the existing **Fin Plate Shear Connection** module within the Osdag framework. The Osdag software (Open Steel Design and Graphics) provides a graphical interface for the design of steel connections as per Indian Standards (IS codes).

The existing module lacked interactive visual feedback for plate capacity and section capacity checks. Specifically, the task required:

- Addition of **Plate Capacity** and **Section Capacity** buttons in the Output Dock of the Fin Plate design interface.
- On clicking these buttons, a **pop-up window** was to be displayed showing:
  - Failure pattern diagrams for **Shear in Plate** and **Tension in Plate**, rendered dynamically according to the user-provided input dimensions.
  - Computed capacity values including Shear Yielding Capacity, Rupture Capacity, Block Shear Capacity, Tension Yielding Capacity, Tension Rupture Capacity, and Axial Block Shear Capacity (in kN).

3.2 Task 1: Tasks Done

1. **Understanding the Osdag Framework** — Studied the existing Osdag codebase, its GUI architecture, and the internal workflow of the **Fin Plate Shear Connection** module. Understood how user-provided input parameters are processed through the design engine and how computed results are rendered in the Output Dock.

2. **Adding Plate Capacity and Section Capacity Buttons** — Identified the appropriate location in the Output Dock panel of the Fin Plate module and added two new interactive buttons — **Plate Capacity** and **Section Capacity** — that become accessible to the user after the design is executed.

3. **Development of the Bolt Pattern Pop-up Dialog** — Developed a dedicated pop-up window that is triggered upon clicking either of the capacity buttons. The pop-up is divided into two sections:

    - **Failure Pattern due to Shear in Plate** — Displays a scaled diagram of the fin plate showing bolt hole positions, edge distances, pitch, and the critical shear rupture path highlighted visually. The following computed values are also displayed alongside the diagram:
        - Shear Yielding Capacity (kN)
        - Rupture Capacity (kN)
        - Block Shear Capacity (kN)
    - **Failure Pattern due to Tension in Plate** — Displays the corresponding tension failure path on the plate diagram along with the following computed values:
        - Tension Yielding Capacity (kN)
        - Tension Rupture Capacity (kN)
        - Axial Block Shear Capacity (kN)

4. **Dynamic Dimension Rendering** — Ensured that all geometric dimensions shown in the failure pattern diagrams — including plate height, bolt edge distances, end

distances, and pitch — update dynamically based on the actual input parameters provided by the user, so that the diagram always corresponds to the specific design under evaluation.

5. **Integration of Computed Capacity Values** — Linked the design engine output directly to the pop-up display, so that all capacity values shown in the dialog are computed in real time based on the current design inputs, ensuring consistency between the numerical results and the visual failure pattern.

6. **Testing and Validation** — Tested the enhanced module with multiple input combinations, varying beam sections, bolt diameters, plate thicknesses, and applied loads, to verify that the diagrams render correctly, dimensions update as expected, and all capacity values are accurately computed and displayed without any errors.

3.3 Task 1: Python Code

This section presents the Python script developed for enhancing the **Fin Plate Shear Connection** module in Osdag. The script implements an interactive pop-up dialog that displays failure pattern diagrams and computed capacity values for both plate capacity and section capacity checks. It uses the `PySide6` framework for GUI rendering and integrates directly with the Osdag design engine to fetch real-time output values.

3.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters**: The script retrieves geometric and design parameters directly from the Osdag connection object, including plate height, plate width, hole diameter, number of bolt rows and columns, plate thickness, bolt spacing, edge distance, end distance, and weld size.

- **Capacity Extraction**: The program fetches computed capacity values from the Osdag design engine output, including Shear Yielding Capacity, Rupture Capacity, Block Shear Capacity, Tension Yielding Capacity, Tension Rupture Capacity, Axial Block Shear Capacity, Moment Demand, and Moment Capacity.

- **Output**: The results are displayed in a resizable, scrollable pop-up dialog containing two panels — a left panel with numerical capacity values and a right panel with dynamically rendered failure pattern diagrams for shear and tension failure modes.

3.3.2 Python Code

The Python script is shown below. Each section is commented for clarity.

Listing 3.1: Fin Plate Capacity Details Dialog — Osdag Enhancement

```
import sys
from PySide6.QtWidgets import (QApplication, QDialog, QWidget,
    QVBoxLayout, QHBoxLayout, QLabel, QGraphicsView, QSizeGrip,
    QGraphicsScene, QScrollArea)
from PySide6.QtCore import Qt
from PySide6.QtGui import (QPainter, QPen, QFont, QColor,
    QPolygonF, QBrush)
from PySide6.QtCore import QPointF
from osdag_gui.ui.components.dialogs.custom_titlebar import
   CustomTitleBar
from osdag_core.Common import *


class FinPlateCapacityDetails(QDialog):
    def __init__(self, connection_obj, rows=3, cols=2, main=None)
       :
        super().__init__()
        app = QApplication.instance()
        self.theme      = app.theme_manager
        self.connection = connection_obj
        self.main       = main

        # Retrieve plate geometry from Osdag connection object
        self.plate_height    = main.plate.height
        self.plate_width     = main.plate.length
        self.hole_dia        = main.bolt.bolt_diameter_provided
```

```
        self.rows            = main.plate.bolts_one_line
        self.cols            = main.plate.bolt_line
        self.plate_thickness = main.plate.thickness

        # Extract capacity values from Osdag output engine
        output      = main.output_values(True)
        dict1       = {i[0]: i[3] for i in output}
        cap_fn      = dict1['button1'][1]
        cap_details = cap_fn(True)
        dd          = {i[1]: i[3] for i in cap_details}

        self.shear_yield_capacity       = float(
            dd['Shear Yielding Capacity (kN)'])
        self.rupture_capacity           = float(
            dd['Rupture Capacity (kN)'])
        self.Block_Shear_Capacity       = float(
            dd['Block Shear Capacity (kN)'])
        self.Tension_Yielding_Capacity  = float(
            dd['Tension Yielding Capacity (kN)'])
        self.Tension_rupture_Capacity   = float(
            dd['Tension Rupture Capacity (kN)'])
        self.axial_block_shear_capacity = float(
            dd['Axial Block Shear Capacity (kN)'])
        self.moment_demand              = float(
            dd['Moment Demand (kNm)'])
        self.moment_capacity            = float(
            dd['Moment Capacity (kNm)'])

        # Organize values into display dictionaries
        self.dict_shear_failure = {
            'Shear Yielding Capacity (kN)': self.
               shear_yield_capacity,
            'Rupture Capacity (kN)':        self.rupture_capacity
               ,
```

```
            'Block Shear Capacity (kN)':    self.
                Block_Shear_Capacity,
        }
        self.dict_tension_failure = {
            'Tension Yielding Capacity (kN)':
                self.Tension_Yielding_Capacity,
            'Tension Rupture Capacity (kN)':
                self.Tension_rupture_Capacity,
            'Axial Block Shear Capacity (kN)':
                self.axial_block_shear_capacity,
        }
        self.weldsize = dict1.get('Weld.Size', 0)
        if not hasattr(self, 'show_third'):
            self.show_third = False
        self.initUI()
```

Explanation of the Code

- **Lines 1–10**: Import necessary libraries including `PySide6` for GUI components, `QPainter`, `QPen`, and `QPolygonF` for custom 2D drawing, and Osdag-specific modules for accessing design output.
- **Lines 13–21**: The `FinPlateCapacityDetails` class is initialized as a `QDialog`. Plate geometry parameters such as height, width, hole diameter, number of bolt rows and columns, and plate thickness are retrieved directly from the Osdag connection object (`main`).
- **Lines 23–44**: Capacity values are extracted from the Osdag design engine by calling `output_values()` and parsing the returned dictionary. Values for shear yielding, rupture, block shear, tension yielding, tension rupture, axial block shear, moment demand, and moment capacity are stored as instance attributes.
- **Lines 46–58**: The extracted values are organized into separate dictionaries (`dict_shear_failure` and `dict_tension_failure`) for structured display in the pop-up panel.

Full Code

The complete Python script developed for the Fin Plate Capacity Details dialog is listed below:

Listing 3.2: Full Code — Fin Plate and Section Capacity Details (fin_plate_capacity.py)

```
import sys
from PySide6.QtWidgets import (QApplication, QDialog, QWidget,
    QVBoxLayout,
                               QHBoxLayout, QLabel, QGraphicsView
                                   , QSizeGrip,
                               QGraphicsScene, QScrollArea)
from PySide6.QtCore import Qt
from PySide6.QtGui import QPainter, QPen, QFont, QColor,
    QPolygonF, QBrush
from PySide6.QtCore import QPointF
from osdag_gui.ui.components.dialogs.custom_titlebar import
    CustomTitleBar
from osdag_core.Common import *


class FinPlateCapacityDetails(QDialog):
    def __init__(self, connection_obj, rows=3, cols=2, main=None)
        :
        super().__init__()
        app = QApplication.instance()
        self.theme      = app.theme_manager
        self.connection = connection_obj
        self.main       = main

        self.plate_height    = main.plate.height
        self.plate_width     = main.plate.length
        self.hole_dia        = main.bolt.bolt_diameter_provided
        self.rows            = main.plate.bolts_one_line
        self.cols            = main.plate.bolt_line
```

```
        self.plate_thickness = main.plate.thickness

        output       = main.output_values(True)
        dict1        = {i[0]: i[3] for i in output}
        cap_fn       = dict1['button1'][1]
        cap_details  = cap_fn(True)
        dd           = {i[1]: i[3] for i in cap_details}

        self.shear_yield_capacity      = float(dd['Shear Yielding
            Capacity (kN)'])
        self.rupture_capacity          = float(dd['Rupture
           Capacity (kN)'])
        self.Block_Shear_Capacity      = float(dd['Block Shear
           Capacity (kN)'])
        self.Tension_Yielding_Capacity = float(dd['Tension
           Yielding Capacity (kN)'])
        self.Tension_rupture_Capacity  = float(dd['Tension
           Rupture Capacity (kN)'])
        self.axial_block_shear_capacity= float(dd['Axial Block
           Shear Capacity (kN)'])
        self.moment_demand             = float(dd['Moment Demand
           (kNm)'])
        self.moment_capacity           = float(dd['Moment
           Capacity (kNm)'])

        self.dict_shear_failure = {
            'Shear Yielding Capacity (kN)': self.
               shear_yield_capacity,
            'Rupture Capacity (kN)':        self.rupture_capacity
               ,
            'Block Shear Capacity (kN)':    self.
               Block_Shear_Capacity,
        }
        self.dict_tension_failure = {
```

```
            'Tension Yielding Capacity (kN)':  self.
                Tension_Yielding_Capacity,
            'Tension Rupture Capacity (kN)':   self.
                Tension_rupture_Capacity,
            'Axial Block Shear Capacity (kN)': self.
                axial_block_shear_capacity,
        }
        self.dict_section_3 = {
            'Moment Demand (kNm)':   self.moment_demand,
            'Moment Capacity (kNm)': self.moment_capacity,
        }

        self.weldsize = 0
        if 'Weld.Size' in dict1:
            self.weldsize = dict1['Weld.Size']

        # Default - child class may override before calling super
            ().__init__()
        if not hasattr(self, 'show_third'):
            self.show_third = False

        self.initUI()

    def setupWrapper(self):
        self.setWindowFlags(Qt.FramelessWindowHint | Qt.
            WindowSystemMenuHint)
        self.setObjectName("spacing_capacity_details")
        ml = QVBoxLayout(self)
        ml.setContentsMargins(1, 1, 1, 1)
        ml.setSpacing(0)
        self.title_bar = CustomTitleBar()
        self.title_bar.setTitle("Bolt Pattern")
        ml.addWidget(self.title_bar)
        self.content_widget = QWidget(self)
```

```
        ml.addWidget(self.content_widget, 1)
        sg = QSizeGrip(self)
        sg.setFixedSize(16, 16)
        ov = QHBoxLayout()
        ov.setContentsMargins(0, 0, 4, 4)
        ov.addStretch(1)
        ov.addWidget(sg, 0, Qt.AlignBottom | Qt.AlignRight)
        ml.addLayout(ov)

    def initUI(self):
        self.setupWrapper()
        sg   = QApplication.primaryScreen().availableGeometry()
        w, h = 900, 500
        self.setGeometry(sg.x() + (sg.width()-w)//2,
                         sg.y() + (sg.height()-h)//2, w, h)

        cl = QVBoxLayout(self.content_widget)
        cl.setContentsMargins(0, 0, 0, 0)
        cl.setSpacing(0)

        sa = QScrollArea()
        sa.setWidgetResizable(True)
        sa.setHorizontalScrollBarPolicy(Qt.ScrollBarAsNeeded)
        sa.setVerticalScrollBarPolicy(Qt.ScrollBarAsNeeded)

        scroll = QWidget()
        scroll.setObjectName("spacing_scroll_widget")
        ml = QHBoxLayout(scroll)
        ml.setContentsMargins(10, 10, 10, 10)

        # left panel
        lp = QWidget()
        lp.setMaximumWidth(400)
        ll = QVBoxLayout()
```

```
        ll.setSpacing(5)

        hl = QLabel("Note: Representative image for Failure
           Pattern (Half Plate)")
        hl.setStyleSheet("font-size: 16px; margin-bottom: 10px;")
        hl.setWordWrap(True)
        ll.addWidget(hl)

        def add_section(title, data):
            lbl = QLabel(title)
            lbl.setStyleSheet("font-size: 14px; font-weight: bold
               ;"
                              "margin-top: 15px; margin-bottom: 5
                                 px;")
            ll.addWidget(lbl)
            for key, val in data.items():
                row = QHBoxLayout()
                row.setContentsMargins(0, 2, 0, 2)
                row.addWidget(QLabel(key))
                row.addStretch()
                v = QLabel(f'{val}')
                v.setStyleSheet("font-size: 12px; font-weight:
                   bold;")
                row.addWidget(v)
                ll.addLayout(row)

        add_section("Failure Pattern due to Shear in Plate",
           self.dict_shear_failure)
        add_section("Failure Pattern due to Tension in Plate",
           self.dict_tension_failure)
        ll.addStretch()
        lp.setLayout(ll)

        # right panel
```

```
        rp = QWidget()
        rl = QVBoxLayout()
        rl.setSpacing(10)
        rp.setLayout(rl)

        def make_view(scene, draw_fn):
            v = QGraphicsView(scene)
            v.setBackgroundBrush(
                QBrush(Qt.white) if self.theme.is_light()
                else QBrush(QColor("#4A4A4A")))
            v.setRenderHint(QPainter.Antialiasing)
            v.setMinimumWidth(500)
            v.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
            v.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
            draw_fn(scene)
            v.fitInView(scene.sceneRect(), Qt.KeepAspectRatio)
            return v

        lb1 = QLabel("Failure Pattern due to Shear in Plate:")
        lb1.setStyleSheet("font-size: 14px; font-weight: bold;
           margin-bottom: 5px;")
        rl.addWidget(lb1)
        self.scene1 = QGraphicsScene()
        self.view1  = make_view(self.scene1, self.createDrawing)
        rl.addWidget(self.view1)

        lb2 = QLabel("Failure Pattern due to Tension in Plate:")
        lb2.setStyleSheet("font-size: 14px; font-weight: bold;"
                          "margin-bottom: 5px; margin-top: 10px;"
                             )
        rl.addWidget(lb2)
        self.scene2 = QGraphicsScene()
        self.view2  = make_view(self.scene2, self.
           createSecondDrawing)
```

```
        rl.addWidget(self.view2)

        # Show third failure pattern only for Beam-Beam
           Connectivity
        if self.show_third:
            lb3 = QLabel("Failure Pattern due to Block Shear in
               Plate:")
            lb3.setStyleSheet("font-size: 14px; font-weight: bold
               ;"
                              "margin-bottom: 5px; margin-top: 10px
                                 ;")
            rl.addWidget(lb3)
            self.scene3 = QGraphicsScene()
            self.view3  = make_view(self.scene3, self.
               createThirdDrawing)
            rl.addWidget(self.view3)

        ml.addWidget(lp, 1)
        ml.addWidget(rp, 2)
        sa.setWidget(scroll)
        cl.addWidget(sa)

    def get_parameters(self):
        param_map = {}
        for item in self.connection.spacing(status=True):
            key, _, _, value = item
            if   key == KEY_OUT_PITCH:     param_map['pitch']  =
               float(value)
            elif key == KEY_OUT_END_DIST:  param_map['end']    =
               float(value)
            elif key == KEY_OUT_GAUGE1:    param_map['gauge1'] =
               float(value)
            elif key == KEY_OUT_GAUGE2:    param_map['gauge2'] =
               float(value)
```

```
            elif key == KEY_OUT_GAUGE:     param_map['gauge']  =
                float(value)
            elif key == KEY_OUT_EDGE_DIST: param_map['edge']   =
                float(value)
        param_map['hole'] = self.main.bolt.bolt_diameter_provided
        return param_map

    def _sc(self, coeff=1):
        p = self.get_parameters()
        s = {
            'pitch':  p['pitch'] / coeff,
            'end':    p['end']   / coeff,
            'edge':   p['edge']  / coeff,
            'width':  self.plate_width  / coeff,
            'height': self.plate_height / coeff,
            'hole':   p['hole']         / coeff,
            'weld':   self.weldsize     / coeff,
        }
        if 'gauge' in p:
            s['g1'] = s['g2'] = p['gauge'] / coeff
        else:
            s['g1'] = p.get('gauge1', 0) / coeff
            s['g2'] = p.get('gauge2', p.get('gauge1', 0)) / coeff
        return s

    def _pens(self, coeff=2):
        out  = QPen(Qt.blue, 2 / coeff)
        if self.theme.is_light():
            dim  = QPen(Qt.black,          1.5 / coeff)
            dash = QPen(Qt.black,          1.5 / coeff, Qt.
                DashLine)
        else:
            dim  = QPen(QColor("#8A8A8A"), 1.5 / coeff)
            dash = QPen(QColor("#8A8A8A"), 1.5 / coeff, Qt.
```

```
            DashLine)
        return out, dim, dash

    def _bxL(self, edge, g1, g2):
        xs, x = [], edge
        for c in range(self.cols):
            xs.append(x)
            if c < self.cols - 1:
                x += g1 if c % 2 == 0 else g2
        return xs

    def _bxR(self, w, edge, g1, g2):
        xs, x = [], w - edge
        for c in range(self.cols):
            xs.append(x)
            if c < self.cols - 1:
                x -= g1 if c % 2 == 0 else g2
        return xs

    def _holes(self, scene, bxs, end, pitch, hole, pen):
        for row in range(self.rows):
            for col in range(self.cols):
                cx = bxs[col]
                cy = end + row * pitch
                scene.addEllipse(cx - hole/2, cy - hole/2, hole,
                    hole, pen)

    def _weld_left(self, scene, weld, h, dim_pen):
        if weld > 0:
            scene.addRect(0, 0, weld, h, QPen(Qt.NoPen), QBrush(
                Qt.red))
            scene.addLine(weld, 0, weld, h, dim_pen)

    def _weld_right(self, scene, weld, w, h, dim_pen):
```

```
        if weld > 0:
            scene.addRect(w - weld, 0, weld, h, QPen(Qt.NoPen),
                QBrush(Qt.red))
            scene.addLine(w - weld, 0, w - weld, h, dim_pen)

    def createDrawing(self, scene):
        coeff = 1
        s = self._sc(coeff)
        outline, dim, dash = self._pens(3)
        w, h  = s['width'], s['height']
        end   = s['end'];   pitch = s['pitch']
        edge  = s['edge'];  g1 = s['g1'];  g2 = s['g2']
        hole  = s['hole'];  weld = s['weld']

        ho, vo = 40/coeff, 60/coeff
        scene.setSceneRect(-ho, -vo, w + 2*vo, h + 2*ho)

        bxs   = self._bxL(edge, g1, g2)
        x_cut = bxs[0]

        scene.addLine(x_cut, end, x_cut, h, dash)
        scene.addLine(x_cut, end, w, end, dash)

        scene.addRect(0, 0, w, h, dim)
        self._holes(scene, bxs, end, pitch, hole, outline)
        self._weld_left(scene, weld, h, dim)

        self._addDimensions(scene, w, h, pitch, end, g1, g2,
                            edge, dim, coeff, mirror=False)

    def createSecondDrawing(self, scene):
        coeff = 1
        s = self._sc(coeff)
        outline, dim, dash = self._pens(3)
```

```
        w, h   = s['width'], s['height']
        end    = s['end'];    pitch = s['pitch']
        edge   = s['edge'];   g1 = s['g1'];   g2 = s['g2']
        hole   = s['hole'];   weld = s['weld']

        ho, vo = 40/coeff, 60/coeff
        scene.setSceneRect(-ho, -vo, w + 2*vo, h + 2*ho)

        bxs    = self._bxL(edge, g1, g2)
        x_cut  = bxs[0]

        scene.addLine(x_cut, end,   w,     end,     dash)
        scene.addLine(x_cut, end,   x_cut, h - end, dash)
        scene.addLine(x_cut, h-end, w,     h - end, dash)

        scene.addRect(0, 0, w, h, dim)
        self._holes(scene, bxs, end, pitch, hole, outline)
        self._weld_left(scene, weld, h, dim)

        self._addDimensions(scene, w, h, pitch, end, g1, g2,
                            edge, dim, coeff, mirror=False)

    def _addDimensions(self, scene, width, height, pitch, end,
                       g1, g2, edge, pen, coeff, mirror):
        ho, vo = 20/coeff, 30/coeff

        if not mirror:
            segs = [(0, edge), (edge, width)]
        else:
            segs = [(0, width - edge), (width - edge, width)]
        for x1, x2 in segs:
            self.addHorizontalDimension(scene, x1, -ho, x2, -ho,
                                        f"{x2-x1:.1f}", pen)
```

```
        self.addVerticalDimension(scene, width+vo, 0,
                                  width+vo, end, str(end), pen)
        for i in range(self.rows - 1):
            self.addVerticalDimension(scene, width+vo, end + i*
               pitch,
                                      width+vo, end + (i+1)*pitch
                                         ,
                                      str(pitch), pen)
        self.addVerticalDimension(scene, width+vo, height,
                                  width+vo, height-end, str(end),
                                      pen)
        total = 2*end + (self.rows-1)*pitch
        self.addVerticalDimension(scene, -vo, 0, -vo, total,
                                  str(total), pen)

    def addHorizontalDimension(self, scene, x1, y1, x2, y2, text,
        pen):
        scene.addLine(x1, y1, x2, y2, pen)
        ext = 10;  arr = 2
        scene.addLine(x1, y1-ext/2, x1, y1+ext/2, pen)
        scene.addLine(x2, y2-ext/2, x2, y2+ext/2, pen)
        fill = (QBrush(Qt.black) if self.theme.is_light()
                else QBrush(QColor("#8A8A8A")))
        for pts in [
            [(x1,y1),(x1+arr,y1-arr/2),(x1+arr,y1+arr/2)],
            [(x2,y2),(x2-arr,y2-arr/2),(x2-arr,y2+arr/2)],
        ]:
            p = scene.addPolygon(
                QPolygonF([QPointF(x, y) for x, y in pts]), pen)
            p.setBrush(fill)
        ti = scene.addText(text)
        f  = QFont(); f.setPointSize(4); ti.setFont(f)
        ti.setDefaultTextColor(Qt.black if self.theme.is_light()
           else Qt.white)
```

```
        if y1 < 0:
            ti.setPos((x1+x2)/2 - ti.boundingRect().width()/2, y1
               -12)
        else:
            ti.setPos((x1+x2)/2 - ti.boundingRect().width()/2, y1
               +5)

    def addVerticalDimension(self, scene, x1, y1, x2, y2, text,
       pen):
        scene.addLine(x1, y1, x2, y2, pen)
        ext = 10;  arr = 2
        scene.addLine(x1-ext/2, y1, x1+ext/2, y1, pen)
        scene.addLine(x2-ext/2, y2, x2+ext/2, y2, pen)
        fill = (QBrush(Qt.black) if self.theme.is_light()
                else QBrush(QColor("#8A8A8A")))
        if y2 > y1:
            polys = [
                [(x1,y1),(x1-arr/2,y1+arr),(x1+arr/2,y1+arr)],
                [(x2,y2),(x2-arr/2,y2-arr),(x2+arr/2,y2-arr)],
            ]
        else:
            polys = [
                [(x2,y2),(x2-arr/2,y2+arr),(x2+arr/2,y2+arr)],
                [(x1,y1),(x1-arr/2,y1-arr),(x1+arr/2,y1-arr)],
            ]
        for pts in polys:
            p = scene.addPolygon(
                QPolygonF([QPointF(x, y) for x, y in pts]), pen)
            p.setBrush(fill)
        ti = scene.addText(text)
        f  = QFont(); f.setPointSize(4); ti.setFont(f)
        ti.setDefaultTextColor(Qt.black if self.theme.is_light()
           else Qt.white)
        if x1 < 0:
```

```
            ti.setPos(x1 - ti.boundingRect().width(),
                      (y1+y2)/2 - ti.boundingRect().height()/2)
        else:
            ti.setPos(x1, (y1+y2)/2 - ti.boundingRect().height()
               /2)


class SectionCapacityDetails(FinPlateCapacityDetails):

    def __init__(self, connection_obj, rows=3, cols=2, main=None,
        show_third=False):
        self.show_third = show_third
        super().__init__(connection_obj, rows, cols, main)

    def createDrawing(self, scene):
        coeff = 1
        s = self._sc(coeff)
        outline, dim, dash = self._pens(3)
        w, h  = s['width'], s['height']
        end   = s['end'];   pitch = s['pitch']
        edge  = s['edge'];  g1 = s['g1'];  g2 = s['g2']
        hole  = s['hole'];  weld = s['weld']

        ho, vo = 40/coeff, 60/coeff
        scene.setSceneRect(-ho, -vo, w + 2*vo, h + 2*ho)

        bxs   = self._bxR(w, edge, g1, g2)
        x_cut = bxs[0]

        scene.addLine(x_cut, end, x_cut, h, dash)
        scene.addLine(0, end, x_cut, end, dash)

        scene.addRect(0, 0, w, h, dim)
        self._holes(scene, bxs, end, pitch, hole, outline)
```

```
        self._weld_right(scene, weld, w, h, dim)

        self._addDimensions(scene, w, h, pitch, end, g1, g2,
                            edge, dim, coeff, mirror=True)

    def createSecondDrawing(self, scene):
        coeff = 1
        s = self._sc(coeff)
        outline, dim, dash = self._pens(3)
        w, h  = s['width'], s['height']
        end   = s['end'];   pitch = s['pitch']
        edge  = s['edge'];  g1 = s['g1'];  g2 = s['g2']
        hole  = s['hole'];  weld = s['weld']

        ho, vo = 40/coeff, 60/coeff
        scene.setSceneRect(-ho, -vo, w + 2*vo, h + 2*ho)

        bxs   = self._bxR(w, edge, g1, g2)
        x_cut = bxs[0]

        scene.addLine(0,     end,   x_cut, end,     dash)
        scene.addLine(x_cut, end,   x_cut, h - end, dash)
        scene.addLine(0,     h-end, x_cut, h - end, dash)

        scene.addRect(0, 0, w, h, dim)
        self._holes(scene, bxs, end, pitch, hole, outline)
        self._weld_right(scene, weld, w, h, dim)

        self._addDimensions(scene, w, h, pitch, end, g1, g2,
                            edge, dim, coeff, mirror=True)

    def createThirdDrawing(self, scene):
        coeff = 1
        s = self._sc(coeff)
```

```
        outline, dim, dash = self._pens(3)
        w, h   = s['width'], s['height']
        end    = s['end'];    pitch = s['pitch']
        edge   = s['edge'];   g1 = s['g1'];   g2 = s['g2']
        hole   = s['hole'];   weld = s['weld']

        ho, vo = 40/coeff, 60/coeff
        scene.setSceneRect(-ho, -vo, w + 2*vo, h + 2*ho)

        bxs    = self._bxR(w, edge, g1, g2)
        x_cut = bxs[0]    # rightmost bolt column x (mirrored)

        last_bolt_y = end + (self.rows - 1) * pitch  # y of last
           bolt row
        scene.addLine(x_cut, 0, x_cut, last_bolt_y, dash)
        scene.addLine(x_cut, last_bolt_y, 0, last_bolt_y, dash)

        scene.addRect(0, 0, w, h, dim)
        self._holes(scene, bxs, end, pitch, hole, outline)
        self._weld_left(scene, weld, h, dim)

        self._addDimensions(scene, w, h, pitch, end, g1, g2,
                            edge, dim, coeff, mirror=True)
```

3.4 Task 1: Documentation

This section presents the contribution towards the Osdag Developer/User Manual, documenting the actual directory structure of the Osdag project and the steps required to run the application.

3.4.1 Directory Structure

The Osdag project follows a modular directory structure as observed in the development environment:

The key components of the directory are described as follows:

- `osdag/` — Core design logic and calculation modules for all steel connection types.
- `osdag_core/` — Contains shared constants, keys, and common utility functions used across all modules, including `Common.py`.
- `osdag_gui/` — Contains all GUI-related components.
  The nested path `ui/components/output_details/` holds individual output dialog scripts for each connection type.
- `fin_plate_c.py` — The script developed during this internship, implementing the `FinPlateCapacityDetails` and `SectionCapacityDetails` dialog classes for the Fin Plate Shear Connection module.
- `bolt_pattern.py` — Handles bolt pattern visualization dialogs for other connection types.
- `ResourceFiles/` — Stores static resources such as section property databases, images, and icons.
- `drawing1.dxf`, `drawing2.dxf` — Auto-generated DXF drawing files produced during design execution.

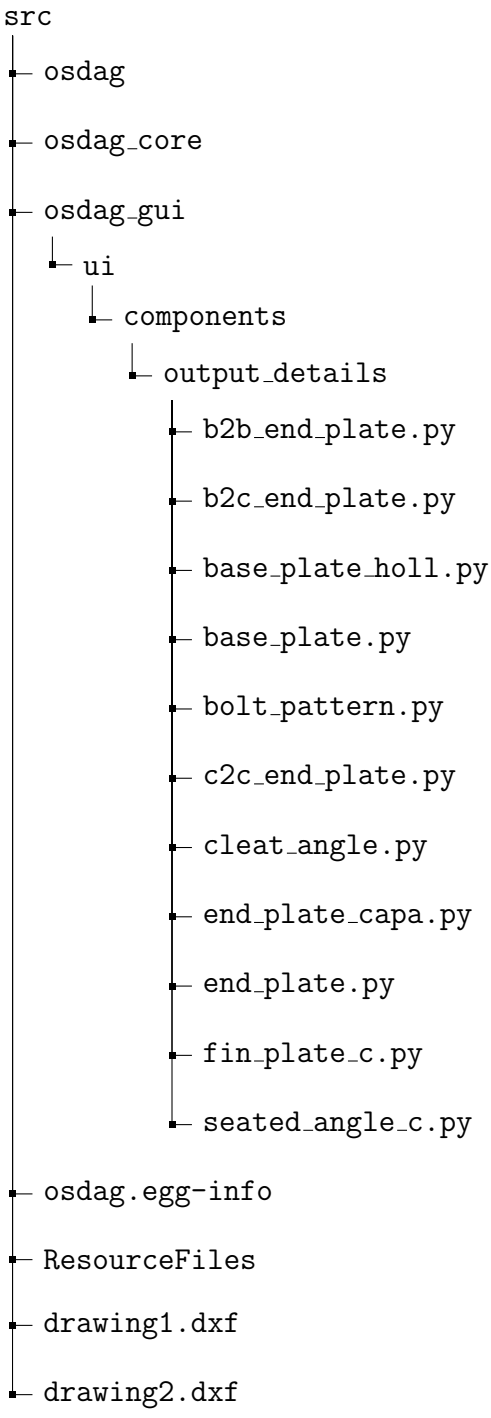

Figure 3.1: Osdag Project Directory Structure

Chapter 4

Internship Task 2 : Enhancement of Shear Connection Capacity Details Module in Osdag

4.1 Task 2: Problem Statement

The second internship task focused on extending the capacity details visualization feature — developed in Task 1 for the Fin Plate module — to three additional shear connection types within the Osdag framework:

- **Cleat Angle Connection**
- **Header Plate Connection**
- **Seated Angle Connection**

Each of these modules lacked interactive visual feedback for capacity checks. The task required implementing dedicated pop-up dialogs for each connection type, displaying dynamically rendered failure pattern diagrams and computed capacity values, consistent with the approach established in Task 1.

Specifically, the task required:

- Addition of **Plate Capacity** and **Section Capacity** buttons in the Output Dock of each respective module.

- On clicking these buttons, a **pop-up window** was to be displayed showing:
  - Dynamically rendered failure pattern diagrams specific to each connection type, updating according to the user-provided input dimensions.
  - Computed capacity values relevant to each connection, including shear, tension, block shear, moment, and section capacity values as applicable.
- For **Cleat Angle**, separate dialogs were required for the supported leg and the supporting leg, each showing different failure patterns and bolt arrangements.
- For **Seated Angle**, the failure patterns covered shear and moment in the angle plate, along with a section capacity view.
- For **Header Plate**, the pop-up displayed failure patterns for shear and tension on both sides of the web, modelling the double-plate nature of the connection.

4.2 Task 2: Tasks Done

The following tasks were performed as part of this internship assignment:

1. **Study of Connection Types** — Studied the structural behaviour and failure modes of Cleat Angle, Header Plate, and Seated Angle shear connections. Understood how each connection transfers load and which failure patterns — shear yielding, tension rupture, block shear, and moment — are relevant to each type.
2. **Cleat Angle — Capacity Details Dialog** — Implemented the `CleatAngleCapacityDetails` class with the following features:
   - Separate dialogs for the **supported leg** (`flag = 0`) and the **supporting leg** (`flag = 1`), each retrieving the appropriate bolt spacing data and capacity values from the Osdag design engine.
   - Failure pattern diagrams for **shear** and **tension**, rendered as paired plate drawings (left and right) with blue bolt holes, dashed failure paths, and dynamic dimension annotations.
   - Display of Shear Capacity, Bearing Capacity, and Bolt Value (kN) in the left information panel.

3. **Cleat Angle — Section Capacity Dialog** — Implemented the `CleatAngleSectionDetails` subclass with:

   - A dedicated section capacity drawing showing the full I-section with both cleat angle plates positioned on either side of the web, bolt holes, weld strips, and dashed failure paths between bolt rows.
   - An optional third drawing for **Block Shear in Section**, rendered when `show_third = True` (activated for Beam-Beam connectivity), showing an L-shaped fracture path on a single plate.
   - Dynamic dimension lines for gauge, end distance, pitch, edge distance, and plate height, all computed from actual design parameters.

4. **Header Plate — Capacity Details Dialog** — Implemented the `EndPlateCapacityDetails` class with:

   - A 3D-perspective style drawing of the I-beam with two end plates on either side of the web, weld strips rendered in orange-brown, and bolt holes in blue.
   - Failure pattern dashed lines for **shear** (vertical cut from plate top to bottom bolt, horizontal to plate edge) and **tension** (U-shaped path between top and bottom bolt rows).
   - Capacity values including Shear Yielding Capacity, Rupture Capacity, Block Shear Capacity, Tension Yielding Capacity, Tension Rupture Capacity, and Axial Block Shear Capacity (kN).

5. **Header Plate — Section Capacity Dialog** — Implemented the `EndPlateSectionDetails` subclass with:

   - A section capacity drawing using the same 3D-style beam geometry with mirrored failure patterns, showing section-side rupture paths.
   - Section capacity values including Supported Section Shear Yielding Capacity, Allowable Shear Capacity, Supporting Section Tension Yielding Capacity, and Section Tension Block Shear Capacity (kN).

6. **Seated Angle — Capacity Details Dialog** — Implemented the `SeatedAngleCapacityDetails` class with:

- Separate drawings for **Failure Pattern due to Shear in Plate** and **Failure Pattern due to Moment in Plate**, each showing the seated angle plate geometry with a shaded top strip (representing the outstanding leg), bolt holes, and dashed failure paths.
- Spacing parameters (end distance, edge distance, gauge, pitch) retrieved directly from bolt attributes of the Osdag connection object.
- Capacity values: Shear Yielding Capacity, Shear Demand, Moment Demand, and Moment Capacity (kN / kNm).

7. **Seated Angle — Section Capacity Dialog** — Implemented the `SeatedAngleSectionDetails` subclass with:
   - A detailed section drawing showing two angle plates (top and bottom), the supported section web between them, bolt holes, inner web lines, and dashed failure paths connecting bolt rows.
   - Dimension annotations for gauge, plate width, plate height, end distance, pitch, and inter-plate clear gap, all rendered dynamically from actual design values.
   - Section capacity values: Supported Section Shear Yielding Capacity, Allowable Shear Capacity, and Supporting Section Tension Yielding Capacity (kN).
8. **Theme Support** — All dialogs implemented full support for both light and dark themes in Osdag, with appropriate pen colors, brush fills, and text colors switching automatically based on `app.theme_manager.is_light()`.
9. **Dynamic Dimension Rendering** — Ensured that all geometric dimensions displayed in every failure pattern diagram update dynamically based on the actual input parameters of the current design, maintaining consistency between the visual output and the numerical results.
10. **Testing and Validation** — Tested all three enhanced modules with multiple input combinations, varying connection types, bolt sizes, angle sections, and applied loads, to verify correct rendering of diagrams, accurate dimension updates, and error-free capacity value display.

4.3 Task 2: Python Code

This section presents the Python scripts developed for the three connection modules enhanced during Task 2. Each script implements the capacity details and section capacity dialog classes, using the `PySide6` framework for GUI rendering and integrating with the Osdag design engine for real-time output values.

4.3.1 Description of the Scripts

- **end_plate_capacity.py** — Implements `EndPlateCapacityDetails` and `EndPlateSectionDetails` for the Header Plate connection. Uses a 3D-perspective beam drawing with double plates on either side of the web.
- **seated_angle_capacity.py** — Implements `SeatedAngleCapacityDetails` and `SeatedAngleSectionDetails` for the Seated Angle connection. Draws the angle plate with shear and moment failure patterns, and a detailed section capacity view.
- **cleat_angle.py** — Implements `CleatAngleDetails`, `CleatAngleCapacityDetails`, and `CleatAngleSectionDetails` for the Cleat Angle connection, with separate handling for supported and supporting legs and an optional block shear pattern for Beam-Beam connectivity.

4.3.2 Code

Header Plate — Capacity Details

The complete source code for the Header Plate Capacity Details dialog is implemented in `end_plate_capacity.py`. It contains the `EndPlateCapacityDetails` and `EndPlateSectionDetails` classes. The file is available in the submitted ZIP file under `codes/end_plate_capacity.py` and in the GitHub repository.

Seated Angle — Capacity Details

The complete source code for the Seated Angle Capacity Details dialog is implemented in `seated_angle_capacity.py`. It contains the `SeatedAngleCapacityDetails` and

`SeatedAngleSectionDetails` classes. The file is available in the submitted ZIP file under `codes/seated_angle_capacity.py` and in the GitHub repository.

Cleat Angle — Capacity Details

The complete source code for the Cleat Angle Capacity Details dialog is implemented in `cleat_anglepy`. It contains the `CleatAngleDetails`, `CleatAngleCapacityDetails`, and `CleatAngleSectionDetails` classes, with separate handling for the supported and supporting legs. The file is available in the submitted ZIP file under `codes/cleat_anglepy` and in the GitHub repository.

4.4 Task 2: Documentation

This section documents the contribution towards the Osdag Developer/User Manual for the three modules enhanced in Task 2.

4.4.1 New Files Added

The following new files were added to the Osdag project under `src/osdag_gui/ui/components/output_details/`:

```
output_details
├─ fin_plate_capacity.py    (Task 1)
├─ end_plate_capacity.py    (Task 2 --- Header Plate)
├─ seated_angle_capacity.py    (Task 2 --- Seated Angle)
└─ cleat_anglepy    (Task 2 --- Cleat Angle)
```

4.4.2 How the Dialogs are Triggered

Each dialog is triggered from the Output Dock of its respective module by clicking the **Plate Capacity** or **Section Capacity** button. The button callback passes the active connection object and the `main` design object to the dialog constructor, which then fetches all required parameters and capacity values from the Osdag design engine automatically.

Listing 4.1: Example: Triggering the Cleat Angle Capacity Dialog

```
from osdag_gui.ui.components.output_details.cleat_angle_capa \
    import CleatAngleCapacityDetails


# flag=0 for supported leg, flag=1 for supporting leg
dialog = CleatAngleCapacityDetails(
    connection_obj=self.connection,
    main=(self, flag)
)
dialog.exec()
```

Chapter 5

Conclusion

5.1 Tasks Accomplished

During the course of this internship with the Osdag team at IIT Bombay under the FOSSEE initiative, the following tasks were successfully completed:

1. **Screening Task — Parametric 3D CAD Model of a Steel Girder Bridge** — Developed a fully parametric, modular 3D CAD model of a short-span steel girder bridge using the `pythonOCC` library in Python. The script `bridge_model.py` assembles I-section girders, a concrete deck slab, and parapets from component factories, supports dynamic parameter adjustment, provides OCC-based visualization, and exports geometry to STEP and BREP formats.

2. **Task 1 — Enhancement of Fin Plate Shear Connection Module** — Enhanced the existing Fin Plate module in Osdag by adding **Plate Capacity** and **Section Capacity** buttons to the Output Dock. Developed the `FinPlateCapacityDetails` and `SectionCapacityDetails` dialog classes, which render dynamic failure pattern diagrams for shear and tension failure modes along with computed capacity values, all updating in real time based on user input parameters.

3. **Task 2 — Enhancement of Shear Connection Capacity Details for Three Additional Modules** — Extended the capacity details visualization feature to three additional shear connection types:

    - **Header Plate** — Implemented `EndPlateCapacityDetails` and

`EndPlateSectionDetails` with 3D-perspective beam drawings, shear and tension failure patterns, and section capacity values.

- **Seated Angle** — Implemented `SeatedAngleCapacityDetails` and `SeatedAngleSectionDetails` with shear and moment failure pattern diagrams, spacing-based dynamic dimensions, and section capacity views.
- **Cleat Angle** — Implemented `CleatAngleCapacityDetails` and `CleatAngleSectionDetails` with separate dialogs for supported and supporting legs, paired plate drawings, and an optional block shear pattern for Beam-Beam connectivity.

4. **Theme and UI Compatibility** — Ensured all developed dialogs support both the light and dark themes of Osdag, with dynamic color switching for pens, brushes, backgrounds, and text throughout all failure pattern diagrams.

5. **Dynamic Dimension Rendering** — Implemented dimension annotation systems across all dialogs that automatically update plate height, bolt edge distances, end distances, pitch, gauge, and total bolt group dimensions based on the active design parameters.

6. **Testing and Validation** — Tested all developed modules across multiple input combinations, connection types, bolt sizes, and applied loads to verify correct rendering, accurate capacity values, and error-free execution.

5.2 Skills Developed

The internship provided significant opportunities to develop both technical and professional skills, as outlined below:

5.2.1 Technical Skills

1. **Python GUI Development with PySide6** — Gained hands-on experience in developing complex, interactive desktop dialog windows using `PySide6`, including `QDialog`, `QGraphicsScene`, `QGraphicsView`, `QScrollArea`, and custom title bars.

2. **2D Technical Drawing with QGraphicsScene** — Developed the ability to programmatically render engineering diagrams including plate outlines, bolt hole patterns, dashed failure paths, weld strips, dimension lines, and arrowheads using `QPen`, `QBrush`, `QPolygonF`, and `QPointF`.

3. **Osdag Framework and Codebase** — Acquired in-depth understanding of the Osdag software architecture, including its design engine, GUI structure, output dock system, and how design parameters flow from user inputs to computed results.

4. **Steel Connection Design** — Strengthened understanding of steel shear connection design as per Indian Standards (IS 800), including failure modes such as shear yielding, shear rupture, block shear, tension yielding, tension rupture, and moment capacity checks for Fin Plate, Cleat Angle, Header Plate, and Seated Angle connections.

5. **3D CAD Modelling with pythonOCC** — Gained experience in parametric 3D solid modelling using the `pythonOCC` (pythonocc-core) library, including extrusion of I-sections, placement of rectangular prisms, geometric transformations, OCC display initialization, and export to STEP and BREP formats.

6. **Object-Oriented Programming** — Strengthened OOP skills through the design of class hierarchies with inheritance, method overriding, and shared utility functions across multiple dialog classes.

7. **Version Control with Git** — Practiced maintaining a structured GitHub repository, committing incremental changes, and organizing code for collaborative open-source development.

5.2.2 Professional Skills

1. **Open Source Contribution** — Gained experience contributing to a real-world open-source engineering software project used by students and professionals across India, understanding the responsibility and standards associated with such work.

2. **Reading and Understanding Existing Codebases** — Developed the ability to navigate, understand, and extend a large, unfamiliar codebase, identifying the right files and functions to modify without breaking existing functionality.

3. **Technical Documentation** — Improved the ability to write clear technical documentation, including code comments, user manual contributions, and structured internship reports.

4. **Problem Solving and Debugging** — Enhanced systematic debugging skills by resolving issues related to dynamic rendering, dimension misalignment, theme compatibility, and parameter extraction from complex design objects.

5. **Time Management and Delivery** — Practiced planning and delivering multiple tasks within internship deadlines, managing the screening task and two internship tasks in parallel with academic commitments.

6. **Communication and Collaboration** — Worked closely with the Osdag project staff, communicated progress and issues effectively, and incorporated feedback to improve the quality of deliverables.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report

Name:	Ayushman Dwivedi	**Project:**	Osdag
Internship:	FOSSEE Semester-Long Internship 2026	**Period:**	16 February 2026 – 16 May 2026

DATE	DAY	TASK	Hours Worked
16-Feb-2026	Sunday	Set up development environment; installed Osdag and studied codebase structure	4
17-Feb-2026	Monday	Studied Osdag GUI architecture: Input Dock, Output Dock, CAD Window, Message Log	5
18-Feb-2026	Tuesday	Reviewed Fin Plate Shear Connection module and existing output dock implementation	5
19-Feb-2026	Wednesday	Studied PySide6 QDialog, QGraphicsScene, QGraphicsView for custom rendering	5
20-Feb-2026	Thursday	Understood IS 800:2007 failure modes: shear yielding, rupture, block shear, tension	5
21-Feb-2026	Friday	Identified locations in Output Dock to add Plate Capacity and Section Capacity buttons	5
22-Feb-2026	Saturday	Studied existing bolt pattern dialogs and connection object data flow in Osdag	4
23-Feb-2026	Sunday	Studied Osdag output dock system and reviewed existing bolt pattern dialog implementations	5
24-Feb-2026	Monday	Explored osdag_core.Common module; understood KEY constants used for spacing and capacity data	5
25-Feb-2026	Tuesday	Added Plate Capacity button to Fin Plate Output Dock; wired callback to dialog trigger	6
26-Feb-2026	Wednesday	Implemented FinPlateCapacityDetails QDialog class skeleton with setupWrapper and initUI	6
27-Feb-2026	Thursday	Developed left info panel showing shear and tension capacity values from design engine	6
28-Feb-2026	Friday	Implemented createDrawing() for Fin Plate shear failure pattern using QGraphicsScene	6
01-Mar-2026	Saturday	Implemented createSecondDrawing() for tension failure pattern with U-shaped dash path	5
02-Mar-2026	Sunday	Integrated _sc() parameter scaling and _pens() theme-aware pen system	5
03-Mar-2026	Monday	Implemented _bxL(), _bxR(), _holes() helpers for bolt column and hole rendering	6
04-Mar-2026	Tuesday	Added weld strip rendering (_weld_left, _weld_right) and linked to design output	6
05-Mar-2026	Wednesday	Implemented _addDimensions() for horizontal and vertical dimension annotations	6
06-Mar-2026	Thursday	Added addHorizontalDimension() and addVerticalDimension() with arrowhead polygons	6
07-Mar-2026	Friday	Implemented SectionCapacityDetails subclass overriding createDrawing with mirrored geometry	7
08-Mar-2026	Saturday	Added createThirdDrawing() for Block Shear failure pattern (Beam-Beam connectivity)	6
09-Mar-2026	Sunday	Integrated light/dark theme support across all Fin Plate dialog components	5
10-Mar-2026	Monday	Tested Fin Plate dialog with multiple bolt configurations and plate dimensions	6
11-Mar-2026	Tuesday	Fixed dimension rendering bugs; verified capacity values match design engine output	6
12-Mar-2026	Wednesday	Reviewed Fin Plate implementation with mentor; noted feedback for improvements	5
13-Mar-2026	Thursday	Incorporated mentor feedback; improved scroll area layout and scene rect padding	6

DATE	DAY	TASK	Hours Worked
14-Mar-2026	Friday	Final testing of Task 1 (fin_plate_c.py); verified error-free execution across inputs	6
15-Mar-2026	Saturday	Reviewed SectionCapacityDetails subclass design; planned mirror geometry for bolt column placement	5
16-Mar-2026	Sunday	Studied Cleat Angle connection structure in IS 800; noted differences from Fin Plate failure modes	5
17-Mar-2026	Monday	Studied Cleat Angle connection: supported vs supporting leg failure modes and bolt layout	5
18-Mar-2026	Tuesday	Implemented CleatAngleDetails class with flag-based supported/supporting leg handling	6
19-Mar-2026	Wednesday	Developed createDrawing() for Cleat Angle bolt pattern with angle strip and holes	6
20-Mar-2026	Thursday	Added gauge1/gauge2 multi-column bolt spacing logic in CleatAngleDetails	6
21-Mar-2026	Friday	Implemented CleatAngleCapacityDetails with paired plate drawings (left + right figures)	7
22-Mar-2026	Saturday	Developed _draw_plate_pair() unified method for shear and tension failure patterns	7
23-Mar-2026	Sunday	Implemented draw_blue_bolt(), add_h_dim(), add_v_dim() helpers in Cleat Angle dialog	6
24-Mar-2026	Monday	Added separate _draw_supporting_leg_shear() and _draw_supporting_leg_tension() methods	7
25-Mar-2026	Tuesday	Implemented CleatAngleSectionDetails with I-section drawing showing web and plate geometry	7
26-Mar-2026	Wednesday	Developed _draw_section_capacity_pattern() with dual cleat plates, bolt holes, weld strips	7
27-Mar-2026	Thursday	Added _draw_section_block_shear_pattern() for optional Block Shear view (Beam-Beam)	6
28-Mar-2026	Friday	Tested Cleat Angle dialogs for both flag=0 and flag=1 with varying bolt configurations	6
29-Mar-2026	Saturday	Fixed dimension misalignment issues in supported leg section drawing	5
30-Mar-2026	Sunday	Began studying Header Plate (End Plate) shear connection structure and failure modes	4
31-Mar-2026	Monday	Implemented EndPlateCapacityDetails with 3D-perspective I-beam and dual plate drawing	7
01-Apr-2026	Tuesday	Developed _draw_common() unified drawing method for shear, tension, and section modes	7
02-Apr-2026	Wednesday	Added 3D-style beam geometry: flanges, web, weld strips in orange-brown, bolt holes in blue	7
03-Apr-2026	Thursday	Implemented shear failure path (vertical cut + horizontal to plate edge) in _draw_common()	6
04-Apr-2026	Friday	Implemented tension failure path (U-shape between top and bottom bolt rows)	6
05-Apr-2026	Saturday	Implemented EndPlateSectionDetails subclass overriding initUI for section capacity view	6
06-Apr-2026	Sunday	Integrated section capacity values from design engine into EndPlateCapacityDetails	6
07-Apr-2026	Monday	Linked dict_section_failure values from supported_section and supporting_section attributes	7
08-Apr-2026	Tuesday	Added addHorizontalDimension() and addVerticalDimension() with fill-brush theming	6
09-Apr-2026	Wednesday	Tested Header Plate dialogs with multiple beam sections and bolt row/column configurations	6
10-Apr-2026	Thursday	Fixed bolt placement and weld strip rendering issues in EndPlateCapacityDetails	5
11-Apr-2026	Friday	Began studying Seated Angle connection: shear and moment failure modes	5
12-Apr-2026	Saturday	Implemented SeatedAngleCapacityDetails with geometry from seated_angle and bolt attributes	7
13-Apr-2026	Sunday	Developed _get_spacing_values() to extract end, edge, gauge, pitch from bolt attributes	6
14-Apr-2026	Monday	Implemented createShearDrawing() with plate, strip, bolt holes and shear failure dash path	7

DATE	DAY	TASK	Hours Worked
15-Apr-2026	Tuesday	Implemented createMomentDrawing() with horizontal + vertical moment failure path	7
16-Apr-2026	Wednesday	Developed _addPlateDimensions() for horizontal and vertical annotations in Seated Angle	6
17-Apr-2026	Thursday	Implemented SeatedAngleSectionDetails with full section drawing showing two angle plates	8
18-Apr-2026	Friday	Developed createSectionDrawing() with web, bolt holes, inner web lines, failure path dashes	7
19-Apr-2026	Saturday	Added dimension annotations for gauge, plate width, plate height, pitch, and inter-plate gap	6
20-Apr-2026	Sunday	Tested Seated Angle dialogs with different rows, cols, and spacing configurations	5
21-Apr-2026	Monday	Fixed bolt centering issues in SeatedAngleSectionDetails section drawing	5
22-Apr-2026	**Tuesday**	**Exam Break**	**0**
23-Apr-2026	**Wednesday**	**Exam Break**	**0**
24-Apr-2026	**Thursday**	**Exam Break**	**0**
25-Apr-2026	**Friday**	**Exam Break**	**0**
26-Apr-2026	**Saturday**	**Exam Break**	**0**
28-Apr-2026	**Monday**	**Exam Break**	**0**
29-Apr-2026	**Tuesday**	**Exam Break**	**0**
30-Apr-2026	**Wednesday**	**Exam Break**	**0**
01-May-2026	Thursday	Full integration test: triggered all capacity dialogs from Output Dock buttons across all four modules	6
02-May-2026	Friday	Verified theme switching (light/dark) across Fin Plate, Cleat Angle, Header Plate, Seated Angle dialogs	6
03-May-2026	Saturday	Resolved scroll area widget sizing issues in CleatAngle and EndPlate dialogs after testing	5
04-May-2026	Sunday	Fixed dynamic dimension updates when users change bolt size, plate thickness, or applied span	5
05-May-2026	Monday	Performed regression testing across all four modules with multiple bolt configurations and plate sizes	6
06-May-2026	Tuesday	Cross-verified all capacity values shown in dialogs against Osdag design engine output	6
07-May-2026	Wednesday	Cleaned up code: removed debug prints, added inline comments, standardized method names	5
08-May-2026	Thursday	Prepared directory structure documentation for Osdag Developer Manual contribution	5
09-May-2026	Friday	Documented how dialogs are triggered from Output Dock with example callback code	5
10-May-2026	Saturday	Drafted Task 1 section of internship report with code explanation and output description	6
11-May-2026	Sunday	Drafted Task 2 section covering Cleat Angle, Header Plate, and Seated Angle modules	6
12-May-2026	Monday	Wrote screening task section: parametric 3D steel girder bridge using pythonOCC	5
13-May-2026	Tuesday	Compiled skills developed section: PySide6, QGraphicsScene, IS 800, OOP, Git	5
14-May-2026	Wednesday	Reviewed full report draft with mentor; incorporated all feedback and corrections	5
15-May-2026	Thursday	Committed all four dialog scripts to GitHub; prepared ZIP with code files and report PDF	4
16-May-2026	Friday	Submitted final internship report to FOSSEE portal; presented work to Osdag team	4

Bibliography

[1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.

[2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.

[3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.