FOSSEE Semester Long Internship - Autumn Report

On

Development of GUI and Web modules of Struts Welded and Bolted to End Gusset for Osdag

Submitted by

Manas Budhiraja

3rd Year B.Tech Student, Department of Civil

National Institute of Technology Karnataka, Surathkal

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

January 26, 2026

Acknowledgments

- I express my sincere gratitude to all those who supported me throughout this fellowship. This journey would not have been possible without the guidance and encouragement of many individuals, and I am deeply thankful for the opportunity to contribute to this project
- 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.
- I would also like to thank my colleagues and fellow interns for the collaborative spirit and for making this a memorable and enriching experience.

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.

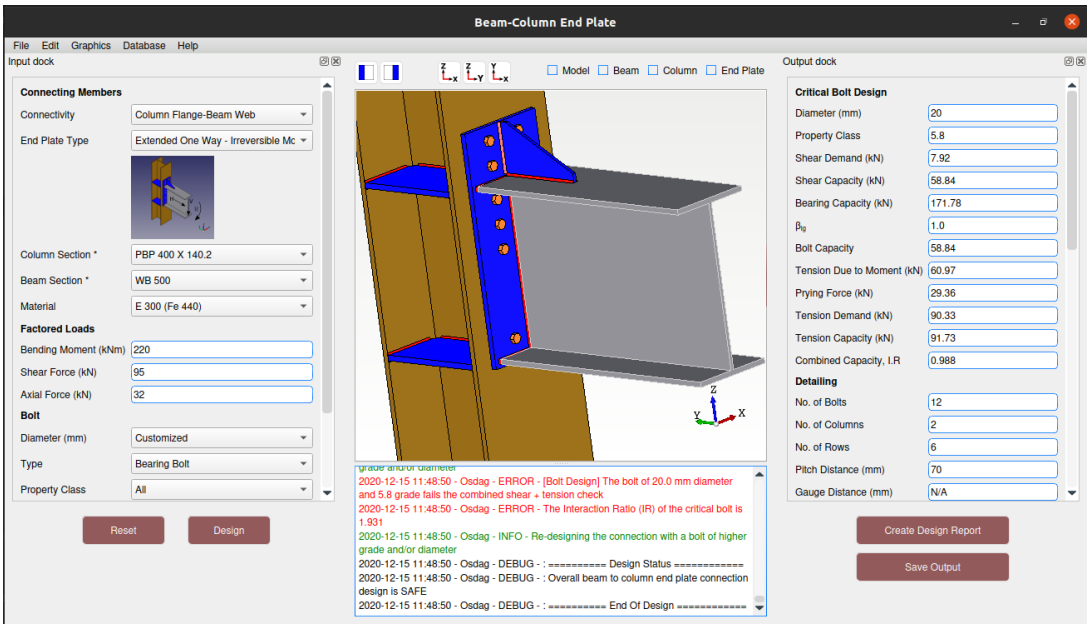


Figure 1.2: Osdag GUI

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 assigned at the beginning of the internship was to extend and improve the existing **Prism Viewer (Shape Viewer App)** into a **complete, modular, and deployable web application** with support for a **plugin-based architecture**. The initial Prism Viewer implementation (desktop + partial web app) demonstrated the core concept of 3D shape rendering, but it had limitations in user experience, UI consistency, and the intended plugin mechanism.

The objective of this task was to build a production-style web application using modern full-stack tools where shapes can be rendered in a browser, and additional shapes can be installed dynamically as plugins. The system needed to support extensibility (via external plugin repositories), containerized deployment (via Docker), and testing/CI automation.

The target tech stack for the screening task was:

- **Frontend:** React.js with Three.js / React-Three-Fiber for 3D rendering
- **Backend:** Django + Django REST Framework (REST APIs)
- **Database:** PostgreSQL
- **Containerization:** Docker + Docker Compose
- **Testing:** Pytest (backend) and Jest/React Testing Library (frontend)

- **CI/CD:** GitHub Actions pipeline for automated test + build

The expected deliverable was a working web app that supports shape selection and rendering, and demonstrates plugin installation using at least one plugin example (**Cylinder**), along with proper documentation and deployment instructions.

2.2 Tasks Done

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

- **Repository Setup and Codebase Analysis:** Studied the existing Prism Viewer repository, identified missing components in the web version, and analysed the architecture gaps related to modular UI and plugin loading.
- **Web Application Development (Frontend):** Developed the Prism Viewer as a React-based web application with an improved UI structure.
  - Implemented a shape rendering interface using **Three.js / React-Three-Fiber**.
  - Designed interactive UI components for selecting shapes and visualizing outputs.
  - Ensured smooth and responsive real-time rendering of 3D objects in the browser.
- **Backend Development (REST APIs):** Built the backend using **Django REST Framework** to support communication between frontend, database, and plugin manager.
  - Designed REST endpoints for fetching available shapes and required metadata.
  - Implemented APIs for plugin installation and retrieval of installed plugins.
  - Added database integration using **PostgreSQL** for storing shape/plugin state.
- **Plugin System Implementation:** Implemented the intended plugin architecture for the web version, matching the desktop concept.
  - Enabled plugins to be maintained in **separate GitHub repositories**.

  - Designed plugin structure containing shape logic, calculations, and rendering methods.
  - Implemented dynamic plugin integration through backend endpoints and frontend updates.
  - Integrated one working plugin example: **Cylinder**.

- **Dockerization and Deployment Automation:** Containerized the complete stack using Docker and Docker Compose.
  - Created Docker configurations for frontend, backend, and PostgreSQL.
  - Ensured the application runs using a single command: `docker-compose up`.
  - Validated the deployment workflow for local testing and demonstration.

- **Testing:** Implemented testing pipelines to validate functionality.
  - Added backend unit tests using **pytest** and Django test utilities.
  - Implemented frontend test cases using **Jest** and **React Testing Library**.
  - Verified core features such as shape rendering and plugin installation flow.

- **CI/CD Pipeline (GitHub Actions):** Configured GitHub Actions workflows for automated checks.
  - Automated test execution on every push to main.
  - Automated Docker image build on successful test runs.

- **Documentation and Demonstration:** Documented setup and usage instructions in `README.md`.
  - Added steps for local setup and Docker-based setup.
  - Added a plugin installation guide for developers and users.
  - Recorded a demo video showing the web app running in a browser, cylinder plugin installation, and Docker execution.
  - **GitHub Repository:**
    `https://github.com/ctheface/Osdag_ManasB`

– **Demo Video (YouTube):**

`https://www.youtube.com/watch?v=EPsbZYAAMDs`

2.2.1 Code Structure

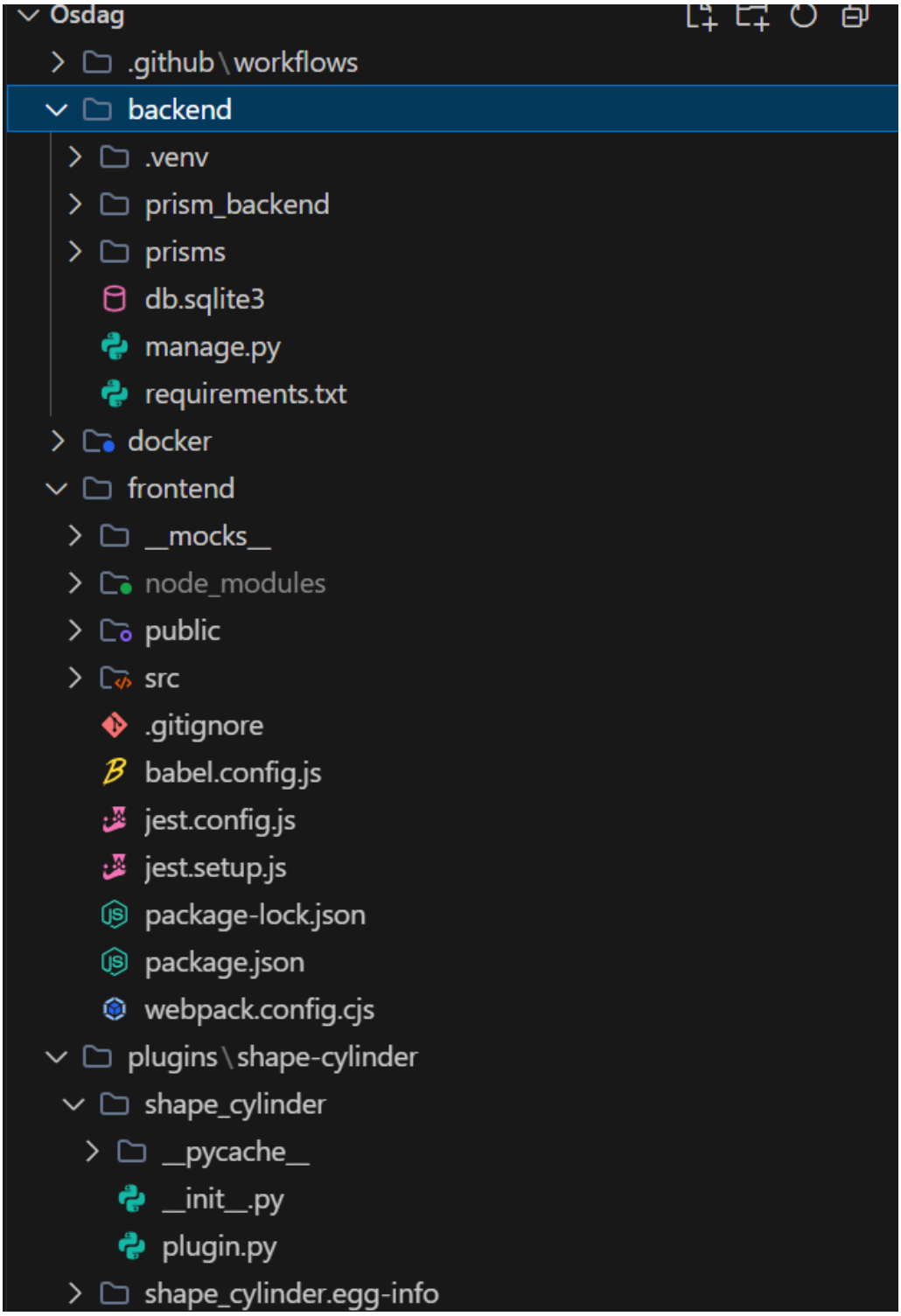


Figure 2.1: Project code structure for Prism Viewer web application.

Chapter 3

Internship Task 1: Compression Members - Struts Welded to End Gusset

3.1 Task 1: Problem Statement

The objective of this task was to upgrade and integrate the **Compression Member module (Strut connection)** for the case where the strut is **welded to an end gusset plate**.

The legacy welded strut module existed in the older Osdag version but required:

- Refactoring of GUI-related workflow to match the updated modular structure,
- Adding missing design computations such as wel, such as beam and column sections and the em Integration of the complete module into the **Osdag Web version**.

3.2 Tasks Done

3.2.1 GUI Refactoring for Welded Strut Module

The GUI module and input-output flow were refactored from the old Osdag implementation into a clean and modular structure suitable for the updated Osdag framework. This included reorganizing UI tabs, input dictionaries, and result rendering logic.

3.2.2 Weld and Gusset Plate Calculation Logic

The welded module was enhanced by implementing proper weld and gusset plate design logic. This ensured realistic output values for strength checks, capacity checks, and pass/fail design status.

3.2.3 Web Integration

The welded-to-end gusset module was integrated into the Osdag web system by aligning:

- Frontend input parameters
- Backend computation calls
- Output result mapping

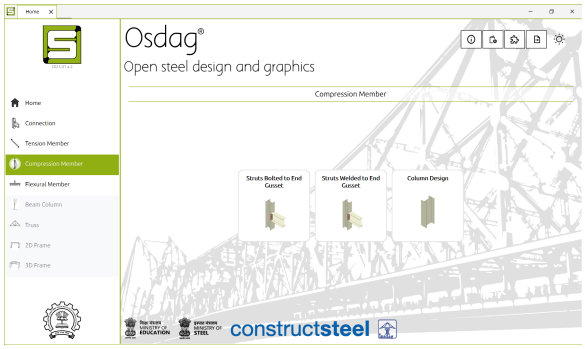


Figure 3.1: Osdag GUI: Compression Member module showing Struts Welded/Bolted to End Gusset.

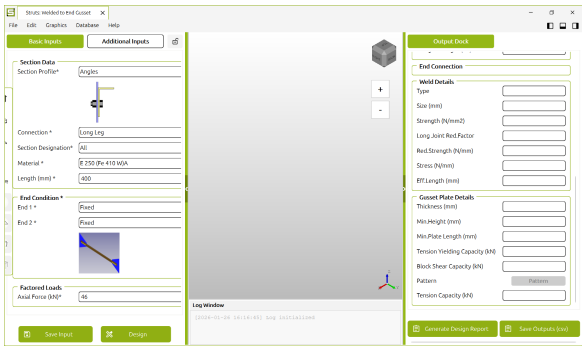


Figure 3.2: Osdag GUI: Welded strut-to-end gusset input and output workflow.

3.3 Task 1: Python Code

This section presents the Python implementation developed for the **Compression Members module** in Osdag, specifically for the case of **struts welded to an end gusset plate**. The implemented logic automates the design workflow by selecting an appropriate gusset plate thickness, designing the weld connection, handling the long joint condition, and verifying the final gusset plate capacity using strength checks such as yielding, rupture, and block shear.

The code is designed to work with the Osdag framework and uses design inputs in the form of a `design_dictionary`, which contains member profile details, connection

location, material parameters, and detailing constraints. The final output includes weld and plate geometry values along with design status messages (*safe/unsafe*) for the welded strut connection.

3.3.1 Description of the Script

The welded strut-to-end gusset design script is structured into the following major steps:

- **Input Parameters:** The design workflow is driven using the `design_dictionary`, which includes key parameters such as: member profile type, location (long leg/short leg), axial compression force, gusset plate thickness list, and material properties.
- **Initial Plate Thickness Selection:** The script begins with an iterative check over the available plate thickness list. Plate material properties are updated per thickness, and the plate is verified for tension yielding and rupture capacity.
- **Weld Selection and Strength Check:** After selecting a suitable plate thickness, the weld size is determined based on member and plate thickness. The weld strength and effective weld length required to resist the applied design force are computed.
- **Long Joint Condition Handling:** The script checks if the computed weld length violates the long joint limit, applies weld strength reduction logic, and adjusts weld configuration where required (especially for channel sections).
- **Final Plate Capacity Verification:** The gusset plate thickness is further validated using yielding, rupture, and block shear capacity checks, ensuring the final design satisfies strength and detailing requirements.

3.3.2 Python Code (Full Implementation)

The complete implementation of the welded strut-to-end gusset workflow is shown below.

Listing 3.1: Strut Welded to End Gusset Plate: Full Plate and Weld Design Logic

```
1 def initial_plate_check(self, design_dictionary):
2         """
3         Initialization of plate thickness based on compression strength
              to determine weld size
4         """
```

```
5           # Use maximum of applied force and 30% of section capacity (
                similar to tension design)
6           # Note: self.load.axial_force is already in Newtons (converted
                in Load class)
7           self.res_force = max((self.load.axial_force), (0.3*self.
                section_capacity))
8
9           self.last_thk = self.plate.thickness[-1]
10
11          for self.plate.thickness_provided in self.plate.thickness:
12              # Update plate material properties for current thickness
13              self.plate.connect_to_database_to_get_fy_fu(grade=self.
                    plate.material,
14                                                          thickness=self.
                                                                plate.
                                                                thickness_provided
                                                                )
15
16              if design_dictionary[KEY_SEC_PROFILE] in ["Channels", 'Back
                    to Back Channels']:
17                  self.plate.tension_yielding(length=self.
                        section_property.depth, thickness=self.plate.
                        thickness_provided,
18                                              fy=self.plate.fy)
19                  self.net_area = self.section_property.depth * self.
                        plate.thickness_provided
20
21              elif design_dictionary[KEY_SEC_PROFILE] == "Star Angles"
                    and design_dictionary[KEY_LOCATION] == 'Long Leg':
22                  self.plate.tension_yielding(length=2*self.
                        section_property.max_leg, thickness=self.plate.
                        thickness_provided,
23                                              fy=self.plate.fy)
24                  self.net_area = 2*self.section_property.max_leg * self.
                        plate.thickness_provided
25
26              elif design_dictionary[KEY_SEC_PROFILE] == "Star Angles"
                    and design_dictionary[KEY_LOCATION] == 'Short Leg':
27                  self.plate.tension_yielding(length=2*self.
```

```
                    section_property.min_leg, thickness=self.plate.
                    thickness_provided,
28                                          fy=self.plate.fy)
29                  self.net_area = 2*self.section_property.min_leg * self.
                    plate.thickness_provided
30
31              else:
32                  if design_dictionary[KEY_LOCATION] == 'Long Leg':
33                      self.plate.tension_yielding(length=self.
                        section_property.max_leg,
34                                                  thickness=self.plate.
                                                    thickness_provided,
                                                    fy=self.plate.fy)
35                      self.net_area = self.section_property.max_leg *
                        self.plate.thickness_provided
36                  else:
37                      self.plate.tension_yielding(length=self.
                        section_property.min_leg,
38                                                  thickness=self.plate.
                                                    thickness_provided,
                                                    fy=self.plate.fy)
39                      self.net_area = self.section_property.min_leg *
                        self.plate.thickness_provided
40
41              self.plate.tension_rupture(A_n=self.net_area, F_u=self.
                plate.fu)
42
43              tension_capacity = min(self.plate.tension_yielding_capacity
                , self.plate.tension_rupture_capacity)
44
45              if tension_capacity > self.res_force:
46                  break
47
48          if design_dictionary[KEY_SEC_PROFILE] in ["Channels", 'Back to
              Back Channels', "Star Angles"]:
49              self.max_tension_yield = 400*self.plate.fy*self.last_thk
                /1.1
50          else:
51              self.max_tension_yield = 200*self.plate.fy*self.last_thk
```

```
                /1.1
52
53          if tension_capacity >= self.res_force:
54              print(f"Plate thickness provided: {self.plate.
                    thickness_provided}")
55              self.thick_design_status = True
56              self.design_status = True
57              self.select_weld(design_dictionary)
58          else:
59              self.design_status = False
60              self.logger.warning(":Compression force {} kN exceeds plate
                    capacity of {} kN for maximum available plate thickness
                    of {} mm.".format(
61                      round(self.res_force / 1000, 2), round(self.
                            max_tension_yield/1000, 2), self.last_thk))
62              self.logger.error(":Design is not safe. \n ")
63              self.logger.info(":==========End Of design===========")
64
65      def select_weld(self, design_dictionary):
66          """
67          Selection of weld size based on the initial thickness
                considered
68          """
69          self.web_weld_status = True
70          if design_dictionary[KEY_SEC_PROFILE] in ["Channels", 'Back to
                Back Channels']:
71              self.thick = self.section_property.web_thickness
72              self.thick_1 = self.section_property.flange_thickness
73          else:
74              self.thick = self.section_property.thickness
75
76          self.weld.weld_size(plate_thickness=self.plate.
                thickness_provided, member_thickness=self.thick, edge_type="
                Rolled")
77
78          self.get_weld_strength(connecting_fu=[self.section_property.fu,
                    self.plate.fu, self.weld.fu],
79                                 weld_fabrication=self.weld.fabrication,
                                       t_weld=self.weld.size, force=(self.
```

```
                                    res_force))
80          print(self.weld.effective, "weld eff")
81          self.weld_plate_length(design_dictionary)
82          self.weld.get_weld_stress(weld_shear=0, weld_axial=self.
                res_force, l_weld=self.weld.length)
83
84          if self.plate.length > (150 * self.weld.throat) and
                design_dictionary[KEY_SEC_PROFILE] in ["Channels", 'Back to
                Back Channels']:
85              self.logger.info(" To satisfy the long joint limit, weld is
                    provided only on the flanges.")
86              self.web_weld_status = False
87              self.weld.weld_size(plate_thickness=self.plate.
                    thickness_provided, member_thickness=self.thick_1,
                    edge_type="Rolled")
88              self.get_weld_strength(connecting_fu=[self.section_property
                    .fu, self.plate.fu, self.weld.fu],
89                                     weld_fabrication=self.weld.
                                           fabrication, t_weld=self.weld.size
                                           ,
90                                     force=(self.res_force))
91              self.weld_plate_length(design_dictionary, "web_weld")
92              self.weld.get_weld_stress(weld_shear=0, weld_axial=self.
                    res_force, l_weld=self.weld.length)
93
94          self.weld.strength_red = self.weld.strength
95          while self.plate.length > (150 * self.weld.throat):
96              self.weld.get_weld_red(t_t=self.weld.throat, strength=self.
                    weld.strength, length=self.plate.length, height=self.
                    plate.height)
97              self.weld.get_weld_stress(weld_shear=0, weld_axial=self.
                    res_force, l_weld=self.weld.length)
98
99              if self.weld.strength_red > self.weld.stress:
100                 self.weld_plate_length(design_dictionary)
101                 break
102             else:
103                 self.weld.effective = round_up((self.res_force/self.
                        weld.strength), 100, 1)
```

```
104                 self.weld_plate_length(design_dictionary)
105
106         if self.weld.strength_red > self.weld.stress:
107             self.weld_design_status = True
108             self.design_status = True
109             self.get_plate_thickness(design_dictionary)
110         else:
111             self.design_status = False
112             self.logger.warning(": The member fails in long joint. \n "
                    )
113             self.logger.error(": Design is unsafe.\n ")
114             self.logger.info(" :==========End Of design===========")
115
116     def get_weld_strength(self, connecting_fu, weld_fabrication, t_weld
        , force, weld_angle=90):
117         """
118         Function to calculate weld strength, effective weld length and
               throat thickness
119         """
120         f_wd = IS800_2007.cl_10_5_7_1_1_fillet_weld_design_stress(
               connecting_fu, weld_fabrication)
121         throat_tk = IS800_2007.
               cl_10_5_3_2_fillet_weld_effective_throat_thickness(t_weld,
               weld_angle)
122         self.Kt = IS800_2007.cl_10_5_3_2_factor_for_throat_thickness(
               weld_angle)
123
124         # Calculate strength per mm of weld (for effective length
               calculation)
125         weld_strength = f_wd * throat_tk
126
127         L_eff = round_up((force/weld_strength), 5, 100)
128
129         self.weld.strength = round(f_wd, 2)
130         self.weld.effective = L_eff
131         self.weld.throat = throat_tk
132
133     def weld_plate_length(self, design_dictionary, web=None):
134         """
```

```
135         Function to calculate weld length, plate length and plate
                height
136         """
137         if design_dictionary[KEY_SEC_PROFILE] == "Channels":
138             if web == None:
139                 self.web_weld = self.section_property.depth - 2 * self.
                        weld.size
140             else:
141                 self.web_weld = 0.0
142             self.flange_weld = round_up(((self.weld.effective - self.
                    web_weld) / 2), 1, 50)
143             self.weld.length = (self.web_weld + 2 * self.flange_weld)
144
145         elif design_dictionary[KEY_SEC_PROFILE] == 'Back to Back
                Channels':
146             if web == None:
147                 self.web_weld = 2 * (self.section_property.depth - 2 *
                        self.weld.size)
148             else:
149                 self.web_weld = 0.0
150             self.flange_weld = round_up(((self.weld.effective - self.
                    web_weld) / 4), 1, 50)
151             self.weld.length = (self.web_weld + 4 * self.flange_weld)
152
153         elif design_dictionary[KEY_SEC_PROFILE] in ["Star Angles",
                Profile_name_2, Profile_name_3] and design_dictionary[
154             KEY_LOCATION] == "Long Leg":
155             if web == None:
156                 self.web_weld = 2 * (self.section_property.max_leg - 2
                        * self.weld.size)
157             else:
158                 self.web_weld = 0.0
159             length_weld = self.section_property.angle_weld_length(self.
                    weld.strength, self.web_weld/2, self.res_force/2,
160                                                                    self.
                                                                        section_proper
                                                                        .
                                                                        Cy
                                                                        ,
```

```
                                                                    self
                                                                    .
                                                                    section_proper
                                                                    .
                                                                    max_leg
                                                                    )
161             self.flange_weld = round_up((length_weld), 1, 50)
162             self.weld.length = (self.web_weld + 4 * self.flange_weld)
163
164         elif design_dictionary[KEY_SEC_PROFILE] in ["Star Angles",
                Profile_name_2, Profile_name_3] and design_dictionary[
165             KEY_LOCATION] == "Short Leg":
166             if web == None:
167                 self.web_weld = 2 * (self.section_property.min_leg - 2
                        * self.weld.size)
168             else:
169                 self.web_weld = 0.0
170             length_weld = self.section_property.angle_weld_length(self.
                    weld.strength, self.web_weld/2, self.res_force/2,
171                                                                 self.
                                                                    section_proper
                                                                    .
                                                                    Cz
                                                                    ,
                                                                    self
                                                                    .
                                                                    section_proper
                                                                    .
                                                                    min_leg
                                                                    )
172             self.flange_weld = round_up((length_weld), 1, 50)
173             self.weld.length = (self.web_weld + 4 * self.flange_weld)
174
175         elif design_dictionary[KEY_SEC_PROFILE] == Profile_name_1 and
                design_dictionary[KEY_LOCATION] == "Long Leg":
176             if web == None:
177                 self.web_weld = (self.section_property.max_leg - 2 *
                        self.weld.size)
178             else:
```

```
179                 self.web_weld = 0.0
180             length_weld = self.section_property.angle_weld_length(self.
                    weld.strength, self.web_weld, self.res_force,
181                                                                    self.
                                                                        section_proper
                                                                        .
                                                                        Cy
                                                                        ,
                                                                        self
                                                                        .
                                                                        section_proper
                                                                        .
                                                                        max_leg
                                                                        )
182             self.flange_weld = round_up((length_weld), 1, 50)
183             self.weld.length = (self.web_weld + 2 * self.flange_weld)
184
185         else:
186             if web == None:
187                 self.web_weld = (self.section_property.min_leg - 2 *
                        self.weld.size)
188             else:
189                 self.web_weld = 0.0
190             length_weld = self.section_property.angle_weld_length(self.
                    weld.strength, self.web_weld, self.res_force,
191                                                                    self.
                                                                        section_proper
                                                                        .
                                                                        Cz
                                                                        ,
                                                                        self
                                                                        .
                                                                        section_proper
                                                                        .
                                                                        min_leg
                                                                        )
192             self.flange_weld = round_up((length_weld), 1, 50)
193             self.weld.length = (self.web_weld + 2 * self.flange_weld)
194
```

```
195         self.plate.length = self.flange_weld + max((4 * self.weld.size)
                , 30)
196         if design_dictionary[KEY_SEC_PROFILE] == "Star Angles" and
                design_dictionary[KEY_LOCATION] == "Long Leg":
197             self.plate.height = 2 * self.section_property.max_leg + max
                    ((4 * self.weld.size), 30)
198         elif design_dictionary[KEY_SEC_PROFILE] == "Star Angles" and
                design_dictionary[KEY_LOCATION] == "Short Leg":
199             self.plate.height = 2 * self.section_property.min_leg + max
                    ((4 * self.weld.size), 30)
200         elif design_dictionary[KEY_SEC_PROFILE] in [Profile_name_1,
                Profile_name_2, Profile_name_3] and design_dictionary[
                KEY_LOCATION] == "Short Leg":
201             self.plate.height = self.section_property.min_leg + max((4
                    * self.weld.size), 30)
202         elif design_dictionary[KEY_SEC_PROFILE] in [Profile_name_1,
                Profile_name_2, Profile_name_3] and design_dictionary[
                KEY_LOCATION] == "Long Leg":
203             self.plate.height = self.section_property.max_leg + max((4
                    * self.weld.size), 30)
204         elif design_dictionary[KEY_SEC_PROFILE] in ['Channels', 'Back
                to Back Channels']:
205             # For Channels, use depth attribute
206             self.plate.height = self.section_property.depth + max((4 *
                    self.weld.size), 30)
207         else:
208             # Default fallback for angles
209             self.plate.height = self.section_property.max_leg + max((4
                    * self.weld.size), 30)
210
211     def get_plate_thickness(self, design_dictionary):
212         """
213         Calculate plate thickness based on the compression capacity
                from the available list of plate thickness
214         """
215         self.plate_last = self.plate.thickness[-1]
216
217         for self.plate.thickness_provided in self.plate.thickness:
218             self.plate.connect_to_database_to_get_fy_fu(grade=self.
```

```
                plate.material,
219                                                     thickness=self.
                                                            plate.
                                                            thickness_provided
                                                            )
220             if design_dictionary[KEY_SEC_PROFILE] in ["Channels", 'Back
                    to Back Channels']:
221                 self.plate.tension_yielding(length=self.plate.height,
222                                             thickness=self.plate.
                                                    thickness_provided, fy=self.
                                                    plate.fy)
223                 self.net_area = (self.plate.height - max((4 * self.weld
                        .size), 30)) * self.plate.thickness_provided
224
225             else:
226                 if design_dictionary[KEY_LOCATION] == 'Long Leg':
227                     self.plate.tension_yielding(length=self.plate.
                            height,
228                                                 thickness=self.plate.
                                                        thickness_provided, fy=
                                                        self.plate.fy)
229                     self.net_area = (self.plate.height - max((4 * self.
                            weld.size), 30)) * self.plate.thickness_provided
230                 else:
231                     self.plate.tension_yielding(length=self.plate.
                            height,
232                                                 thickness=self.plate.
                                                        thickness_provided, fy=
                                                        self.plate.fy)
233                     self.net_area = (self.plate.height - max((4 * self.
                            weld.size), 30)) * self.plate.thickness_provided
234
235             self.plate.tension_rupture(A_n=self.net_area, F_u=self.
                    plate.fu)
236
237             A_vg = (self.plate.length - max((2 * self.weld.size), 15))
                    * self.plate.thickness_provided
238             A_vn = A_vg
239             A_tg = (self.plate.height - max((2 * self.weld.size), 15))
```

```
                * self.plate.thickness_provided
240             A_tn = A_tg
241
242             self.plate.tension_blockshear_area_input(A_vg=A_vg, A_vn=
                    A_vn, A_tg=A_tg, A_tn=A_tn,
243                                                      f_u=self.plate.fu,
                                                             f_y=self.plate.
                                                             fy)
244
245             self.plate_tension_capacity = min(self.plate.
                    tension_yielding_capacity,
246                                               self.plate.
                                                      tension_rupture_capacity
                                                      ,
247                                               self.plate.
                                                      block_shear_capacity)
248
249             print(self.plate.tension_yielding_capacity, self.plate.
                    tension_rupture_capacity, self.plate.
                    block_shear_capacity)
250
251             if self.plate_tension_capacity > self.res_force:
252                 self.design_status = True
253                 break
254
255             elif (self.plate_tension_capacity < self.res_force) and
                    self.plate.thickness_provided == self.plate_last:
256                 self.design_status = False
257                 self.logger.warning(": The factored compression force
                        ({} kN) exceeds the plate capacity of {} kN with
                        respect to the maximum available "
258                                     "plate thickness of {} mm.".format(
259                         round(self.res_force / 1000, 2), round(self.
                            max_tension_yield/1000, 2), self.plate_last)
                            )
260                 self.logger.error(":Design is unsafe. \n ")
261                 self.logger.info(":=========End Of design==========")
262             else:
263                 pass
```

```
264
265         if self.plate_tension_capacity > self.res_force:
266             if (2 * self.plate.length) > self.length:
267                 self.design_status = False
268                 self.logger.warning(":The plate length of {} mm is
                        higher than the member length of {} mm".format(
269                     2 * self.plate.length, self.length))
270                 self.logger.info(":Try a larger weld size and/or
                        increase the member length.")
271                 self.logger.error(":Design is unsafe. \n ")
272                 self.logger.info(":=========End Of design==========")
273             else:
274                 self.plate_design_status = True
275                 self.design_status = True
276
277                 self.logger.info(":Overall welded compression member
                        design is safe. \n")
278                 self.logger.info(" :=========End Of design==========")
279         else:
280             self.design_status = False
281             self.logger.error(": Design is not safe. \n ")
282             self.logger.info(" :=========End Of design==========")
```

3.3.3 Explanation of the Code

- `initial_plate_check()`: Iterates over available plate thicknesses and selects the first thickness that satisfies the governing design force using plate yielding and rupture capacity checks.
- `select_weld()`: Determines weld size based on plate/member thickness, computes required weld length, checks weld stress, and handles long joint condition logic.
- `get_weld_strength()`: Calculates weld design stress and effective throat thickness as per IS 800:2007 provisions and determines the effective weld length required for the applied force.
- `weld_plate_length()`: Computes weld distribution and plate geometry (length and height) based on the selected profile type and connection location.

- `get_plate_thickness()`: Performs final gusset plate verification using yielding, rupture, and block shear capacities and generates the final safe/unsafe design status.

3.4 Task 1: Documentation

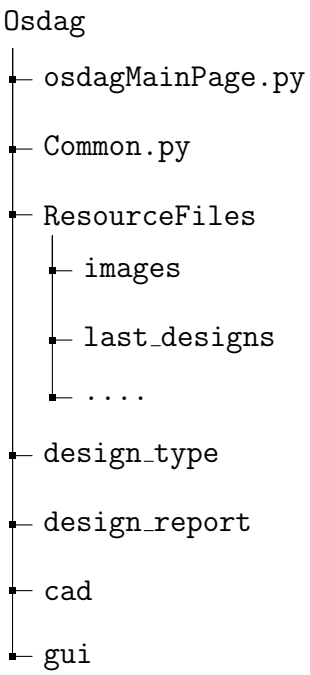
As part of this internship task, documentation was prepared to support both **developers** and **end users** for the newly implemented and enhanced compression member modules. The documentation was written in alignment with the Osdag Developer/User Manual format and includes workflow explanations, module integration details, and usage instructions for the implemented connection types.

3.4.1 Documentation Contents

The documentation for this task covers:

- Overview of compression member design for struts connected to end gusset plates
- Welded-to-end gusset workflow and output interpretation
- Bolted-to-end gusset workflow and output interpretation
- Input parameters and design preference options
- Output fields, design checks, and pass/fail status reporting
- Module integration notes for GUI and web implementation

3.4.2 Directory Structure



Program Start Calls The main entry point for the program is `osdagMainPage.py`. To start the program, open the Osdag folder and start the terminal with that path, execute the following command from the project root:

```
$ python osdagMainPage.py
```

Chapter 4

Internship Task 2: Compression Members – Struts Bolted to End Gusset Plate

4.1 Task 2: Problem Statement

The objective of this task was to develop the **Compression Members module for struts bolted to an end gusset plate** in Osdag. Unlike the welded-to-end gusset case (which required refactoring of existing legacy logic), this bolted module was implemented **from scratch**. The scope included complete design workflow development, covering input handling, member checks, bolted connection detailing, and structured output generation.

Additionally, the module was integrated into:

- the refactored Osdag GUI workflow, and
- the Osdag web version,

ensuring that design inputs, computation flow, and output reporting remain consistent across both platforms.

4.2 Task 2: Tasks Done

The bolted strut-to-end gusset module was implemented as a complete workflow. The major tasks completed are summarised below.

4.2.1 Module Development from Scratch

A new bolted connection module was developed to support compression struts connected to an end gusset plate. The work included:

- Defining design inputs and UI field mapping for bolted end gusset connections
- Implementing section selection and slenderness-based validation for compression members
- Implementing bolt design checks (bolt sizing, capacity, and detailing constraints)
- Implementing gusset plate sizing and verification checks
- Generating structured output results compatible with Osdag Output Dock and report generation

4.2.2 Design Checks and Workflow Implementation

The design workflow implemented in this task includes:

- Member selection based on compression strength and slenderness limits
- Effective length calculation based on end conditions (End 1 / End 2)
- Bolt shear and bolt bearing capacity evaluation
- Detailing checks such as spacing, edge distance, and applicable reduction factors
- Plate checks (yielding, rupture, and block shear) to finalise the safe/unsafe design status

4.2.3 Integration into Refactored GUI and Web Version

The developed bolted module was integrated into the refactored Osdag GUI and the Osdag web workflow by ensuring:

- consistent input dictionary mapping,
- dynamic UI updates through Osdag callbacks,

- correct output mapping into Output Dock widgets,
- compatibility with the Osdag report generation structure.

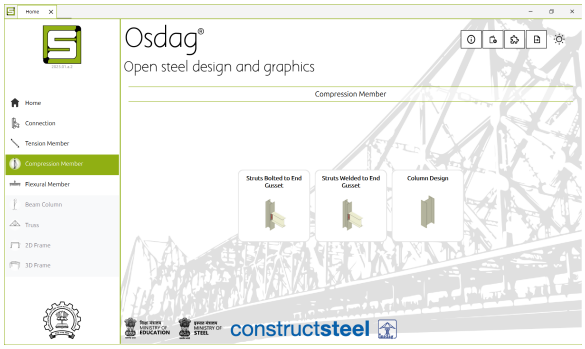


Figure 4.1: Osdag GUI: Compression Member module showing Struts Welded/Bolted to End Gusset.

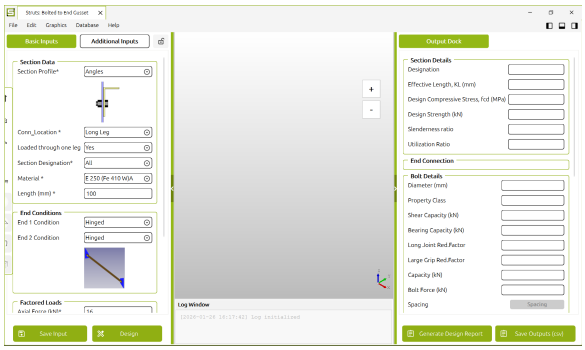


Figure 4.2: Osdag GUI: Bolted strut-to-end gusset input and output workflow.

4.3 Task 2: Python Code

This section presents the Python code developed for **struts bolted to end gusset plate**. The implementation follows Osdag's class-based module structure and includes handling of design preferences, input processing, design calculations, and output formatting.

4.3.1 Code File Included

The complete module logic is implemented in:

- `compression_bolted.py`

4.3.2 Python Code Listing (Key Portion)

To avoid long compilation times, the main design workflow implementation is included below as a code listing.

Listing 4.1: Strut Bolted to End Gusset Plate: Main Module Implementation (`compression_bolted.py`)

```
1 """
2 Started on 28th December, 2025.
3
4 @author: Manas Budhiraja
```

```
5
6   Module: Compression Member Bolted Design (Struts Bolted to End Gusset)
7
8   Reference:
9               1) IS 800: 2007 General construction in steel - Code of
                   practice (Third revision)
10              2) Design of Steel Structures by N. Subramanian (Fifth
                   impression, 2019)
11  """
12
13  from ..member import Member
14  from ...Common import *
15  from ...utils.common.component import *
16  from ...utils.common.common_calculation import *
17  from ...utils.common.load import Load
18  from ...utils.common.Section_Properties_Calculator import
       BBAngle_Properties, SAngle_Properties, BBChannel_Properties
19  from ...utils.common.material import *
20  from ...Report_functions import *
21  from ...design_report.reportGenerator_latex import CreateLatex
22  from ...custom_logger import CustomLogger
23  from ...utils.common import is800_2007
24  from ...utils.common.is800_2007 import IS800_2007
25  from pylatex.utils import NoEscape
26  from pathlib import Path
27  from importlib.resources import files
28  import math
29  import numpy as np
30  import logging
31  import sys
32  import os
33  import os
34  import shutil
35  import time
36  import sys
37
38
39  class Compression_bolted(Member):
40
```

```
41      def __init__(self):
42          print(f'Entering Compression_bolted')
43          super(Compression_bolted, self).__init__()
44          self.design_status = False
45          self.hover_dict = {}
46
47      def module_name(self):
48          return KEY_DISP_STRUT_BOLTED_END_GUSSET
49
50      def set_osdaglogger(self, key):
51          """
52          Function to set Logger for Compression Bolted Module
53          """
54          # Set Custom logger
55          logging.setLoggerClass(CustomLogger)
56
57          # Create unique logger name per instance
58          unique_logger_name = '
                Osdag_struts_bolted_end_gusset_compress_member'
59          self.logger = logging.getLogger(unique_logger_name)
60
61          if not isinstance(self.logger, CustomLogger):
62              logging.getLogger(unique_logger_name).manager.loggerDict.
                    pop(unique_logger_name, None)
63              self.logger = logging.getLogger(unique_logger_name)
64
65          # Clear any existing handlers
66          self.logger.handlers.clear()
67          self.logger.setLevel(logging.DEBUG)
68
69          # Shared formatter for all handlers
70          formatter = logging.Formatter(
71              fmt='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
72              datefmt='%Y-%m-%d %H:%M:%S'
73          )
74
75          # ---------- CONSOLE HANDLER ----------
76          console_handler = logging.StreamHandler()
77          console_handler.setFormatter(formatter)
```

```
78          self.logger.addHandler(console_handler)
79
80          # ---------- FILE HANDLER (CLEAR & RESTART LOG) ----------
81          log_dir = Path("ResourceFiles") / "logs"
82          log_dir.mkdir(parents=True, exist_ok=True)
83          log_file_path = log_dir / f"{unique_logger_name}.log"
84
85          file_handler = logging.FileHandler(
86              log_file_path,
87              mode="w",           # clears previous log
88              encoding="utf-8"
89          )
90          file_handler.setFormatter(formatter)
91          self.logger.addHandler(file_handler)
92
93          # ---------- GUI HANDLER ----------
94          if key is not None:
95              gui_handler = OurLog(key)
96              gui_handler.setFormatter(formatter)
97              self.logger.addHandler(gui_handler)
98
99          # Initialize components for the design
100         self.plate = Plate(thickness=[0.0], material_grade="E 250 (Fe
                410 W)A")
101         self.bolt = Bolt(grade=[0.0], diameter=[0.0], bolt_type="",
                bolt_hole_type="Standard",
102                          edge_type="Sheared or hand flame cut", mu_f
                             =0.3, corrosive_influences=True)
103
104     def tab_list(self):
105         """
106         :return: This function returns the list of tuples. Each tuple
                will create a tab in design preferences, in the
107         order they are appended. Format of the Tuple is:
108         [Tab Title, Type of Tab, function for tab content)
109         """
110         tabs = []
111         t1 = (DISP_TITLE_ANGLE, TYPE_TAB_1, self.tab_angle_section)
112         tabs.append(t1)
```

```
113         t2 = (DISP_TITLE_CHANNEL, TYPE_TAB_1, self.tab_channel_section)
114         tabs.append(t2)
115         t6 = ("Connector", TYPE_TAB_2, self.plate_connector_values)
116         tabs.append(t6)
117         t3 = ("Bolt", TYPE_TAB_2, self.bolt_values)
118         tabs.append(t3)
119         t4 = ("Detailing", TYPE_TAB_2, self.detailing_values)
120         tabs.append(t4)
121         t5 = ("Design", TYPE_TAB_2, self.design_values)
122         tabs.append(t5)
123         return tabs
124
125     def tab_channel_section(self, input_dictionary):
126         """Override parent method to handle non-numeric plate thickness
                values"""
127         # Check if plate thickness is a valid number before calling
                parent
128         if input_dictionary and KEY_PLATETHK in input_dictionary:
129             plate_thk_data = input_dictionary[KEY_PLATETHK]
130             if isinstance(plate_thk_data, list) and len(plate_thk_data)
                    > 0:
131                 try:
132                     float(plate_thk_data[0])
133                 except (ValueError, TypeError):
134                     # If plate thickness is not numeric, set a default
                            value
135                     input_dictionary[KEY_PLATETHK] = [10.0]  # Default
                            plate thickness
136
137         # Call parent implementation
138         return super().tab_channel_section(input_dictionary)
139
140     def tab_angle_section(self, input_dictionary):
141         """Override parent method to handle non-numeric plate thickness
                values"""
142         # Check if plate thickness is a valid number before calling
                parent
143         if input_dictionary and KEY_PLATETHK in input_dictionary:
144             plate_thk_data = input_dictionary[KEY_PLATETHK]
```

```
145             if isinstance(plate_thk_data, list) and len(plate_thk_data)
                    > 0:
146                 try:
147                     float(plate_thk_data[0])
148                 except (ValueError, TypeError):
149                     # If plate thickness is not numeric, set a default
                        value
150                     input_dictionary[KEY_PLATETHK] = [10.0]  # Default
                        plate thickness
151
152         # Call parent implementation
153         return super().tab_angle_section(input_dictionary)
154
155
156     def tab_value_changed(self):
157         """
158         :return: This function is used to update the values of the keys
                in design preferences,
159          which are dependent on other inputs.
160         """
161         change_tab = []
162
163         t1 = (DISP_TITLE_ANGLE, [KEY_SECSIZE, KEY_SEC_MATERIAL, '
            Label_0'],
164               [KEY_SECSIZE_SELECTED, KEY_SEC_FY, KEY_SEC_FU, 'Label_1',
                    'Label_2', 'Label_3', 'Label_4', 'Label_5',
165                'Label_7', 'Label_8', 'Label_9',
166                'Label_10', 'Label_11', 'Label_12', 'Label_13', '
                   Label_14', 'Label_15', 'Label_16', 'Label_17',
167                'Label_18',
168                'Label_19', 'Label_20', 'Label_21', 'Label_22', '
                   Label_23', 'Label_24', KEY_IMAGE], TYPE_TEXTBOX,
169               self.get_new_angle_section_properties)
170         change_tab.append(t1)
171
172         t2 = (DISP_TITLE_ANGLE, ['Label_1', 'Label_2', 'Label_3','
            Label_0'],
173               ['Label_7', 'Label_8', 'Label_9', 'Label_10', 'Label_11',
                    'Label_12', 'Label_13', 'Label_14', 'Label_15',
```

```
174                 'Label_16', 'Label_17', 'Label_18', 'Label_19', '
                        Label_20', 'Label_21', 'Label_22', 'Label_23',
175                 KEY_IMAGE],
176               TYPE_TEXTBOX, self.get_Angle_sec_properties)
177         change_tab.append(t2)
178
179         t3 = (DISP_TITLE_CHANNEL, [KEY_SECSIZE, KEY_SEC_MATERIAL,'
                Label_0'],
180               [KEY_SECSIZE_SELECTED, KEY_SEC_FY, KEY_SEC_FU, 'Label_1',
                    'Label_2', 'Label_3', 'Label_13', 'Label_14',
181                'Label_4', 'Label_5',
182                'Label_9', 'Label_10', 'Label_11', 'Label_12', 'Label_15
                    ', 'Label_16', 'Label_17',
183                'Label_19', 'Label_20', 'Label_21',
184                'Label_22', 'Label_23', 'Label_26','Label_27', KEY_IMAGE
                    ], TYPE_TEXTBOX, self.
                    get_new_channel_section_properties)
185         change_tab.append(t3)
186
187         t4 = (DISP_TITLE_CHANNEL, ['Label_1', 'Label_2', 'Label_3', '
                Label_13','Label_14'],
188               ['Label_9', 'Label_10','Label_11', 'Label_12', 'Label_15'
                    , 'Label_16', 'Label_17','Label_19', 'Label_20', '
                    Label_21', 'Label_22','Label_26','Label_27', KEY_IMAGE
                    ], TYPE_TEXTBOX, self.get_Channel_sec_properties)
189         change_tab.append(t4)
190
191         t5 = ("Connector", [KEY_CONNECTOR_MATERIAL], [KEY_CONNECTOR_FU,
                KEY_CONNECTOR_FY_20, KEY_CONNECTOR_FY_20_40,
192                                                       KEY_CONNECTOR_FY_40
                                                            ],
                                                            TYPE_TEXTBOX,
                                                            self.get_fu_fy
                                                            )
193         change_tab.append(t5)
194
195         t6 = (DISP_TITLE_ANGLE, [KEY_SECSIZE_SELECTED], [KEY_SOURCE],
                TYPE_TEXTBOX, self.change_source)
196         change_tab.append(t6)
```

```
197
198         t7 = (DISP_TITLE_CHANNEL, [KEY_SECSIZE_SELECTED], [KEY_SOURCE],
                TYPE_TEXTBOX, self.change_source)
199         change_tab.append(t7)
200
201         return change_tab
202
203     def input_dictionary_design_pref(self):
204         """
205         :return: This function is used to choose values of design
                preferences to be saved to design dictionary.
206         """
207         design_input = []
208
209         # Use actual tab names like tension_bolted.py does
210         t1 = (DISP_TITLE_ANGLE, TYPE_COMBOBOX, [KEY_SEC_MATERIAL])
211         design_input.append(t1)
212
213         t1a = (DISP_TITLE_CHANNEL, TYPE_COMBOBOX, [KEY_SEC_MATERIAL])
214         design_input.append(t1a)
215
216         t2 = ("Bolt", TYPE_COMBOBOX, [KEY_DP_BOLT_TYPE,
                KEY_DP_BOLT_HOLE_TYPE, KEY_DP_BOLT_SLIP_FACTOR])
217         design_input.append(t2)
218
219         t4 = ("Detailing", TYPE_COMBOBOX, [KEY_DP_DETAILING_EDGE_TYPE,
                KEY_DP_DETAILING_CORROSIVE_INFLUENCES])
220         design_input.append(t4)
221
222         t5 = ("Detailing", TYPE_TEXTBOX, [KEY_DP_DETAILING_GAP])
223         design_input.append(t5)
224
225         t6 = ("Design", TYPE_COMBOBOX, [KEY_DP_DESIGN_METHOD])
226         design_input.append(t6)
227
228         t7 = ("Connector", TYPE_COMBOBOX, [KEY_CONNECTOR_MATERIAL])
229         design_input.append(t7)
230
231         return design_input
```

```
232
233     def input_dictionary_without_design_pref(self):
234         """
235         :return: This function is used to choose values of design
                preferences to be saved to
236         design dictionary if design preference is never opened by user.
237         """
238         design_input = []
239
240         t1 = (KEY_MATERIAL, [KEY_SEC_MATERIAL], 'Input Dock')
241         design_input.append(t1)
242
243         t2 = (None, [KEY_DP_BOLT_TYPE, KEY_DP_BOLT_HOLE_TYPE,
                KEY_DP_BOLT_SLIP_FACTOR,
244                      KEY_DP_DETAILING_EDGE_TYPE,
                         KEY_DP_DETAILING_CORROSIVE_INFLUENCES,
                         KEY_DP_DETAILING_GAP,
245                      KEY_DP_DESIGN_METHOD, KEY_CONNECTOR_MATERIAL], '')
246         design_input.append(t2)
247
248         return design_input
249
250     def refresh_input_dock(self):
251         """
252         :return: This function returns list of tuples which has keys
                that needs to be updated,
253          on changing Keys in design preference (ex: adding a new
                section to database should reflect in input dock)
254         """
255         add_buttons = []
256
257         t2 = (DISP_TITLE_ANGLE, KEY_SECSIZE, TYPE_COMBOBOX_CUSTOMIZED,
                KEY_SECSIZE_SELECTED, KEY_SEC_PROFILE,
258               VALUES_SEC_PROFILE_2, Profile_name_1)
259         add_buttons.append(t2)
260
261         return add_buttons
262
263
```

```
264     def fn_profile_section(self, arg=None):
265         if arg is None or len(arg) == 0:
266             return []
267         profile = arg[0]
268         # Return appropriate section sizes based on profile type
269         if profile in ['Angles', 'Back to Back Angles', 'Star Angles']:
270             return connectdb("Angles", call_type="popup")
271         elif profile in ['Channels', 'Back to Back Channels']:
272             return connectdb("Channels", call_type="popup")
273         return []
274 
275     def fn_conn_type(self, args):
276         """Function to populate location based on the type of section""
            "
277         if args is None or len(args) == 0:
278             return VALUES_LOCATION_1
279         conn = args[0]
280         if conn in ['Angles', 'Back to Back Angles', 'Star Angles']:
281             return VALUES_LOCATION_1
282         elif conn in ["Channels", "Back to Back Channels"]:
283             return VALUES_LOCATION_2
284         return VALUES_LOCATION_1
285 
286     def fn_conn_image(self, args):
287         if args is None or len(args) == 0:
288             return VALUES_IMG_TENSIONBOLTED[0]
289         img = args[0]
290         if img == VALUES_SEC_PROFILE_2[0]: # Angles
291             return VALUES_IMG_TENSIONBOLTED[0]
292         elif img == VALUES_SEC_PROFILE_2[1]: # Back to Back Angles
293             return VALUES_IMG_TENSIONBOLTED[1]
294         elif img == VALUES_SEC_PROFILE_2[2]: # Star Angles
295             return VALUES_IMG_TENSIONBOLTED[2]
296         elif img == VALUES_SEC_PROFILE_2[3]: # Channels
297             return VALUES_IMG_TENSIONBOLTED[3]
298         else:
299             return VALUES_IMG_TENSIONBOLTED[4]
300 
301     def out_bolt_bearing(self, args):
```

```
302         """Returns True to hide bolt bearing output when bolt type is
                not bearing"""
303         bolt_type = args[0]
304         if bolt_type != TYP_BEARING:
305             return True
306         else:
307             return False
308
309     def fn_end1_end2(self, arg):
310         """Function to populate End 2 options based on End 1 selection"
                ""
311         end1 = arg[0]
312         if end1 == 'Fixed':
313             return VALUES_STRUT_END2
314         elif end1 == 'Free':
315             return ['Fixed']
316         elif end1 == 'Hinged':
317             return ['Fixed', 'Hinged']
318         elif end1 == 'Roller':
319             return ['Fixed', 'Hinged']
320         return VALUES_STRUT_END2
321
322     def fn_end1_image(self, arg):
323         """Function to return image path based on End 1 condition"""
324         if arg == 'Fixed':
325             return str(files("osdag_core.data.ResourceFiles.images").
                    joinpath("RRRRstrut.png"))
326         elif arg == 'Free':
327             return str(files("osdag_core.data.ResourceFiles.images").
                    joinpath("RRRRstrut.png"))
328         elif arg == 'Hinged':
329             return str(files("osdag_core.data.ResourceFiles.images").
                    joinpath("RRRFstrut.png"))
330         elif arg == 'Roller':
331             return str(files("osdag_core.data.ResourceFiles.images").
                    joinpath("RRRRstrut.png"))
332         return str(files("osdag_core.data.ResourceFiles.images").
                joinpath("RRRRstrut.png"))
333
```

```
334     def fn_end2_image(self, arg):
335         """Function to return image path based on End 1 and End 2
                conditions"""
336         end1 = arg[0]
337         end2 = arg[1]
338 
339         if end1 == 'Fixed':
340             if end2 == 'Fixed':
341                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RRRRstrut.png"))
342             elif end2 == 'Free':
343                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RRRRstrut.png"))
344             elif end2 == 'Hinged':
345                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RFRFstrut.png"))
346             elif end2 == 'Roller':
347                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RRRRstrut.png"))
348         elif end1 == 'Free':
349             return str(files("osdag_core.data.ResourceFiles.images").
                    joinpath("RRRRstrut.png"))
350         elif end1 == 'Hinged':
351             if end2 == 'Fixed':
352                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RRRFstrut.png"))
353             elif end2 == 'Hinged':
354                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RFRFstrut.png"))
355             elif end2 == 'Roller':
356                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RRRRstrut.png"))
357         elif end1 == 'Roller':
358             if end2 == 'Fixed':
359                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RRRRstrut.png"))
360             elif end2 == 'Hinged':
361                 return str(files("osdag_core.data.ResourceFiles.images"
                        ).joinpath("RRRRstrut.png"))
```

```
362
363         return str(files("osdag_core.data.ResourceFiles.images").
                joinpath("RRRRstrut.png"))
364
365     def out_intermittent(self, args):
366         """Returns True to hide intermittent connection fields for
                single sections"""
367         sec_type = args[0]
368         if sec_type in ['Back to Back Angles', 'Star Angles', 'Back to
                Back Channels']:
369             return False
370         else:
371             return True
372
373     def customized_input(self):
374         """Function to populate combobox based on the option selected""
                "
375         c_lst = []
376
377         t1 = (KEY_SECSIZE, self.fn_profile_section)
378         c_lst.append(t1)
379         t2 = (KEY_GRD, self.grdval_customized)
380         c_lst.append(t2)
381         t3 = (KEY_D, self.diam_bolt_customized)
382         c_lst.append(t3)
383         t4 = (KEY_PLATETHK, self.plate_thick_customized)
384         c_lst.append(t4)
385
386         return c_lst
387
388     def input_values(self, existingvalues={}):
389         '''
390         Function to return a list of tuples to be displayed as the UI.(
                Input Dock)
391         '''
392         self.module = KEY_DISP_STRUT_BOLTED_END_GUSSET
393         self.mainmodule = 'Member'
394
395         options_list = []
```

```
396
397         t1 = (KEY_MODULE, KEY_DISP_STRUT_BOLTED_END_GUSSET, TYPE_MODULE
                , None, True, 'No Validator')
398         options_list.append(t1)
399
400         t1 = (None, KEY_SECTION_DATA, TYPE_TITLE, None, True, 'No
                Validator')
401         options_list.append(t1)
402
403         t2 = (KEY_SEC_PROFILE, KEY_DISP_SEC_PROFILE, TYPE_COMBOBOX,
                VALUES_SEC_PROFILE_2, True, 'No Validator')
404         options_list.append(t2)
405
406         t3 = (KEY_IMAGE, None, TYPE_IMAGE, VALUES_IMG_TENSIONBOLTED[0],
                 True, 'No Validator')
407         options_list.append(t3)
408
409         t3 = (KEY_LOCATION, KEY_DISP_LOCATION, TYPE_COMBOBOX,
                VALUES_LOCATION_1, True, 'No Validator')
410         options_list.append(t3)
411
412         # New Input: Loaded through one leg
413         t_load = ('is_leg_loaded', 'Loaded through one leg',
                TYPE_COMBOBOX, ['Yes', 'No'], True, 'No Validator')
414         options_list.append(t_load)
415
416         t4 = (KEY_SECSIZE, KEY_DISP_SECSIZE, TYPE_COMBOBOX_CUSTOMIZED,
                ['All','Customized'], True, 'No Validator')
417         options_list.append(t4)
418
419         t4 = (KEY_MATERIAL, KEY_DISP_MATERIAL, TYPE_COMBOBOX,
                VALUES_MATERIAL, True, 'No Validator')
420         options_list.append(t4)
421
422         t5 = (KEY_LENGTH, KEY_DISP_LENGTH, TYPE_TEXTBOX, None, True, '
                Int Validator')
423         options_list.append(t5)
424
425         # New Inputs: End Conditions
```

```
426         t9 = (None, 'End Conditions', TYPE_TITLE, None, True, 'No
                Validator')
427         options_list.append(t9)
428
429         t10 = (KEY_END1, 'End 1 Condition', TYPE_COMBOBOX,
                VALUES_STRUT_END1, True, 'No Validator')
430         options_list.append(t10)
431
432         t11 = (KEY_END2, 'End 2 Condition', TYPE_COMBOBOX,
                VALUES_STRUT_END2, True, 'No Validator')
433         options_list.append(t11)
434
435         t12 = (KEY_IMAGE_two, None, TYPE_IMAGE_COMPRESSION, str(files("
                osdag_core.data.ResourceFiles.images").joinpath("RRRRstrut.
                png")), True, 'No Validator')
436         options_list.append(t12)
437
438         t7 = (None, DISP_TITLE_FSL, TYPE_TITLE, None, True, 'No
                Validator')
439         options_list.append(t7)
440
441         t8 = (KEY_AXIAL, KEY_DISP_AXIAL_STAR, TYPE_TEXTBOX, None, True,
                 'Int Validator')
442         options_list.append(t8)
443
444         t8 = (None, DISP_TITLE_BOLT, TYPE_TITLE, None, True, 'No
                Validator')
445         options_list.append(t8)
446
447         t10 = (KEY_D, KEY_DISP_D, TYPE_COMBOBOX_CUSTOMIZED, VALUES_D,
                True, 'No Validator')
448         options_list.append(t10)
449
450         t11 = (KEY_TYP, KEY_DISP_TYP, TYPE_COMBOBOX, VALUES_TYP, True,
                'No Validator')
451         options_list.append(t11)
452
453         t12 = (KEY_GRD, KEY_DISP_GRD, TYPE_COMBOBOX_CUSTOMIZED,
                VALUES_GRD, True, 'No Validator')
```

```
454         options_list.append(t12)
455
456         t13 = (None, DISP_TITLE_PLATE, TYPE_TITLE, None, True, 'No
                Validator')
457         options_list.append(t13)
458
459         t14 = (KEY_PLATETHK, KEY_DISP_PLATETHK,
                TYPE_COMBOBOX_CUSTOMIZED, VALUES_PLATETHK, True, 'No
                Validator')
460         options_list.append(t14)
461
462         return options_list
463
464     def safe_log(self, level, message):
465         """
466         Safely log a message, catching RuntimeError if the Qt widget
                handler
467         has been deleted (e.g., log window was closed).
468         """
469         try:
470             if level == 'info':
471                 self.logger.info(message)
472             elif level == 'warning':
473                 self.logger.warning(message)
474             elif level == 'error':
475                 self.logger.error(message)
476         except RuntimeError:
477             # Qt widget handler was deleted - ignore silently
478             pass
479
480
481     def output_values(self, flag):
482         '''
483         Function to return a list of tuples to be displayed as the UI.(
                Output Dock)
484         '''
485         out_list = []
486
487         t1 = (None, DISP_TITLE_STRUT_SECTION, TYPE_TITLE, None, True)
```

```
488         out_list.append(t1)
489
490         t2 = (KEY_DESIGNATION, KEY_DISP_DESIGNATION, TYPE_TEXTBOX,
491               self.section_size_1.designation if flag else '', True)
492         out_list.append(t2)
493
494         # Effective Length (KL) - Calculated as K * L
495         t_eff = ('KEY_EFFECTIVE_LENGTH', 'Effective Length, KL (mm)',
              TYPE_TEXTBOX,
496               self.effective_length if flag else '', True)
497         out_list.append(t_eff)
498
499         # Design Compressive Stress (fcd) - IS 800 Cl 7.1.2.1
500         t_fcd = ('KEY_FCD', 'Design Compressive Stress, fcd (MPa)',
              TYPE_TEXTBOX,
```

4.4 Task 2: Code Logic Workflow (Flow Explanation)

The design logic for the **struts bolted to end gusset plate** module follows a structured pipeline, where each stage performs a specific design validation and updates the final design outputs. The complete workflow is implemented in the file `compression_bolted.py`.

4.4.1 Flowchart-Style Workflow

The overall execution flow of the module is summarised below:

1. **UI Input Collection (Input Dock)**
    - User selects the section profile type (Angles / Built-up Angles / Channels / Built-up Channels).
    - User provides member properties such as section size, material grade, member length, axial compression load, and connection details.
    - End conditions (End 1 and End 2) are selected to determine the effective length factor.

2. **Input Validation**

   - Basic validation ensures that mandatory inputs such as member length and axial compression force are non-zero and correctly defined.
   - If errors are detected, the design workflow terminates and warnings are displayed to the user.

3. **Initialisation of Components and Parameters**

   - Member section, plate object, and bolt object are initialised using Osdag component classes.
   - Design preferences such as bolt type, bolt hole type, detailing edge type, and design method are mapped into the design dictionary.
   - Safety factors γ_{m0} and γ_{m1} are assigned as per IS 800:2007.

4. **Effective Length Calculation (End Condition Based)**

   - The effective length factor K is computed based on the selected End 1 and End 2 conditions.
   - The effective length KL is then used for slenderness evaluation and member strength calculations.

5. **Member Capacity and Slenderness Check**

   - For each available section size, the slenderness ratio is computed using:

$$\lambda = \frac{KL}{r_{\min}}$$

   - Sections that fail the slenderness limit are rejected.
   - Design compressive stress f_{cd} is computed using IS 800 compressive stress formulation.
   - Compression capacity is computed and compared against the required design axial compression force.
   - The module selects the most suitable section satisfying both slenderness and strength requirements.

6. **Bolt Selection and Connection Design**

   - Bolt diameter is selected based on minimum spacing and geometric feasibility.
   - The governing connection force is computed as:

$$P_{res} = \max\left(P_u,\ 0.3 \times P_{member}\right)$$

   - Bolt shear and bolt bearing capacities are evaluated (based on bolt type).
   - Reduction factors such as long joint and large grip effects are applied where required.
   - The final bolt arrangement and spacing parameters are generated.

7. **Gusset Plate Design and Strength Checks**

   - Plate thickness is selected from the available thickness list.
   - Plate dimensions (height and length) are computed based on bolt pattern, pitch, edge distance, and detailing rules.
   - The plate is checked for:
     - tension yielding capacity,
     - tension rupture capacity, and
     - block shear capacity,

     and the minimum safe capacity governs the design.

8. **Final Output Rendering**

   - Output Dock values are populated with member results (designation, slenderness, efficiency, compression capacity).
   - Connection outputs include bolt capacities, reduction factors, bolt force, spacing details, and plate capacities.
   - The final status is reported as **safe** or **unsafe**.

4.5 Task 2: Summary of Work

This task involved the complete development and integration of the **compression members module for struts bolted to an end gusset plate** in Osdag. Unlike the welded connection case, this module was implemented entirely from scratch and required establishing the full design workflow, including input processing, member checks, connection detailing, and output generation.

The implementation covers compression member selection based on strength and slenderness limits, effective length handling through end condition inputs, bolt capacity and detailing checks, and gusset plate verification using yielding, rupture, and block shear criteria. The final design status is reported clearly as safe or unsafe along with governing design values.

In addition to the core design logic, the module was successfully integrated into both the refactored Osdag GUI and the Osdag web version. Care was taken to ensure consistent input mapping, seamless UI interaction, accurate output rendering in the Output Dock, and compatibility with the existing Osdag report generation framework.

Chapter 5

Conclusions

5.1 Tasks Accomplished

During the internship, the work was primarily focused on developing and integrating compression member connection modules in Osdag. The tasks completed are summarised below.

5.1.1 Task 1: Struts Welded to End Gusset Plate

- **Refactoring of Legacy Welded Module**
  - Refactored the existing welded-to-end gusset code from the older Osdag version into a cleaner and modular structure.
  - Improved separation of input handling, computation flow, and output generation for maintainability.
  - Updated the GUI module flow to align with the refactored Osdag framework.
- **Implementation of Weld Design Logic**
  - Implemented weld size selection logic based on member and gusset plate thickness.
  - Computed effective weld length, weld throat thickness, and weld stress checks using IS 800:2007 provisions.
  - Added long joint condition handling and weld strength reduction checks where applicable.

- **Implementation of Gusset Plate Design Checks**
  - Implemented plate thickness selection from available thickness options based on capacity checks.
  - Verified gusset plate against tension yielding, rupture, and block shear criteria.
  - Computed final gusset plate geometry (length and height) based on weld requirements and detailing rules.
- **Web Integration of Welded Module**
  - Mapped frontend inputs to backend welded design logic for the Osdag web version.
  - Ensured computed results were correctly rendered in the web output dock and report sections.
  - Validated end-to-end execution flow for real-time design output generation.
- **Testing and Output Verification**
  - Verified the welded module output values and safe/unsafe status through multiple input combinations.
  - Ensured consistency between GUI outputs and web outputs after integration.

5.1.2 Task 2: Struts Bolted to End Gusset Plate

- **Development of Bolted Module from Scratch**
  - Created the full bolted strut-to-end gusset design workflow without relying on legacy implementation.
  - Implemented input definitions, design preference mapping, and structured computation flow.
  - Generated final output formatting compatible with Osdag output and reporting structure.
- **Connection Design Logic and Checks**

  - Implemented bolt capacity and detailing checks (bolt arrangement, spacing, edge distance, and governing force).
  - Performed member selection and verification using compression capacity and slenderness-based validation.
  - Implemented gusset plate verification checks (yielding, rupture, and block shear) to finalise safe/unsafe status.

- **Integration into Refactored GUI and Web Version**
  - Integrated the bolted module into the refactored Osdag GUI input-output workflow.
  - Integrated the module into the Osdag web version with consistent input and output mapping.
  - Verified correct output rendering in the Output Dock and ensured compatibility with report generation.

5.2 Skills Developed

During the internship, the following technical and professional skills were developed and strengthened:

- **Structural Design Automation:** Gained hands-on experience in implementing automated steel connection design workflows based on IS 800:2007 provisions.
- **Python Development:** Improved ability to write and refactor Python code for large-scale engineering applications, including modular function design and structured computation flow.
- **GUI Refactoring and Integration:** Developed experience in refactoring legacy GUI workflows into cleaner, maintainable modules while ensuring compatibility with updated application architecture.
- **Web Integration:** Learned how to integrate backend design logic with the Osdag web version by mapping inputs, executing design computations, and rendering structured outputs.

- **Debugging and Error Handling:** Strengthened debugging skills for complex design modules through iterative testing, log analysis, and resolving integration-related issues.
- **Code Maintainability and Modularity:** Developed an understanding of writing maintainable engineering software using modular design patterns and clear separation of responsibilities.
- **Technical Documentation:** Enhanced ability to document engineering workflows and software modules in a structured format suitable for developer/user manuals and final reports.

Chapter A

Appendix

A.1 Internship Work Report

Name:	Manas Budhiraja
Project:	Osdag
Internship:	FOSSEE Semester Long Internship - Autumn 2025

The internship work was carried out in a structured and iterative manner, starting with environment setup and repository familiarisation, followed by module development, testing, and integration. Initially, time was dedicated to studying the Osdag codebase, understanding the existing compression member workflow, and analysing legacy implementations to plan the required refactoring and enhancements. The development phase involved implementing missing design logic, integrating the modules into the refactored GUI and Osdag web version, and validating results through repeated test cases. The internship was conducted on a consistent schedule of approximately **4 hours per working day** (**Monday to Friday**), with occasional work on weekends. The work log below summarises the major activities completed during the period **1st October 2025 to 31st December 2025**.

	Day	Task	Hours
1st Oct 2025 -- 3rd Oct 2025	Wed--Fri	• Setup Osdag development environment and dependencies• Studied Osdag repository structure and module architecture• Understood existing compression member workflows and UI docking pattern	12
6th Oct 2025 -- 10th Oct 2025	Mon--Fri	• Analysed legacy logic for welded strut-to-end gusset module• Identified computation flow and key design checks (plate + weld)• Planned refactoring strategy for cleaner separation of functions	26
13th Oct 2025 -- 17th Oct 2025	Mon--Fri	• Refactored legacy welded module into modular structure• Improved readability, maintainability, and input-output mapping• Updated GUI workflow alignment with refactored structure	17
20th Oct 2025 -- 24th Oct 2025	Mon--Fri	• Implemented weld selection logic (weld size and fabrication handling)• Computed weld strength, throat thickness, and effective length• Added weld stress evaluation as per IS 800:2007 provisions	22
27th Oct 2025 -- 31st Oct 2025	Mon--Fri	• Implemented long joint condition checks and weld strength reduction• Implemented weld length distribution across plate geometry• Validated safe/unsafe weld status using stress vs strength checks	23
3rd Nov 2025 -- 7th Nov 2025	Mon--Fri	• Implemented gusset plate thickness selection logic• Added plate capacity checks: yielding, rupture, block shear• Finalised plate geometry output (height and length)	20
10th Nov 2025 -- 14th Nov 2025	Mon--Fri	• Integrated welded strut module into Osdag web version• Connected input mapping to backend computation workflow• Verified output rendering in output dock and report values	19
17th Nov 2025 -- 21st Nov 2025	Mon--Fri	• Performed testing and output verification for welded module• Ensured consistency between GUI outputs and web outputs• Fixed integration issues and ensured stable module execution	23
24th Nov 2025 -- 28th Nov 2025	Mon--Fri	• Started Task 2: bolted strut-to-end gusset module from scratch• Defined inputs, design preferences mapping, and workflow structure• Implemented initial member checks and effective length handling	26
1st Dec 2025 -- 5th Dec 2025	Mon--Fri	• Implemented bolt design checks (shear, bearing, and reductions)• Developed bolt arrangement logic with spacing and detailing constraints• Generated structured connection outputs for output dock rendering	16
8th Dec 2025 -- 12th Dec 2025	Mon--Fri	• Implemented gusset plate design checks for bolted module• Added yielding, rupture, and block shear verification• Finalised safe/unsafe status workflow and reporting logic	20
15th Dec 2025 -- 19th Dec 2025	Mon--Fri	• Integrated bolted module into refactored Osdag GUI• Ensured correct UI interaction and output dock updates• Verified correctness of output mappings and computed values	18
22nd Dec 2025 -- 26th Dec 2025	Mon--Fri	• Integrated bolted module into Osdag web version• Ensured consistent input mapping and output reporting• Performed stability testing for end-to-end execution	21

	Day	Task	Hours
29th Dec 2025 -- 31st Dec 2025	Mon--Wed	• Final testing, bug fixing, and output verification for Task 2• Cleaned module structure and ensured readiness for submission• Prepared final report content, screenshots, and code references	9
Total		Total Hours	272 hrs

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.