FOSSEE Autumn Semester Internship Report

On

Integration of Compression Force Support in Osdag Connection Modules

Submitted by

Nishi Kant Mandal

4th Year B.Tech Student, Department of Civil Engineering

B.I.T. Sindri

Sindri, Dhanbad, Jharkhand

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Parth Karia

Ajmal Babu M S

Ajinkya Dahale

December 22, 2025

Acknowledgments

I would like to extend my sincere gratitude to the following individuals and organizations whose guidance and support were instrumental in the successful completion of this fellowship project:

- My heartfelt thanks go to the Osdag team, particularly **Ajmal Babu M. S.**, **Ajinkya Dahale**, and **Parth Karia**, for their continuous technical expertise and collaborative support during the development of the Welded Butt and Lap Joint modules.
- I am truly grateful for the mentorship and inspiring leadership of **Prof. Siddhartha Ghosh**, Principal Investigator of the Osdag project, from the Department of Civil Engineering at IIT Bombay.
- I would like to express my gratitude to **Prof. Kannan M. Moudgalya**, Principal Investigator of the FOSSEE project, and the entire FOSSEE team in the Department of Chemical Engineering at IIT Bombay for their invaluable project guidance and academic support.
- My sincere appreciation is extended to **Usha Viswanathan**, **Vineeta Parmar**, and the FOSSEE management team for their exceptional administrative support, which ensured the seamless progression of this internship.
- I gratefully acknowledge the financial and institutional support from the **National Mission on Education through Information and Communication Technology (NMEICT)**, Ministry of Education, Government of India.
- Finally, I would like to thank my alma mater, **B.I.T. Sindri**, along with the mentors and administrative staff who have supported me 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.

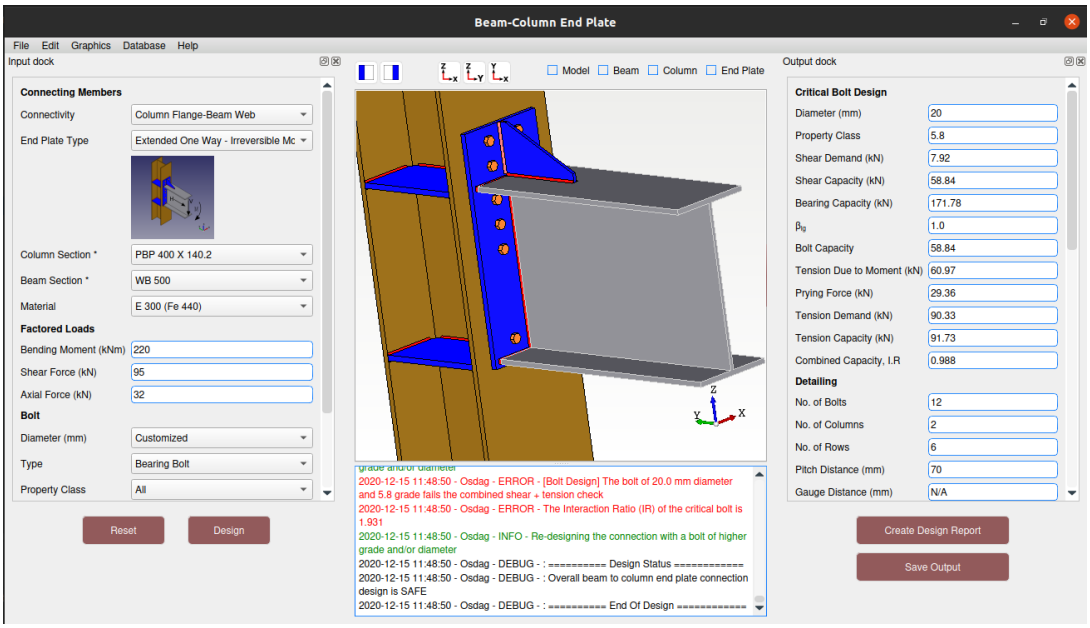

Figure 1.2: Osdag GUI

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

Internship Task 1: Integration of Compression Force Support in Welded Connection Modules

This module was developed by Nishi Kant Mandal (`https://github.com/nishikantmandal007/Osdag/tree/feat/Compression_force`) during the fellowship period.

2.1 Task 1: Problem Statement

The primary objective of this task was to extend the existing **Welded Butt Joint** and **Welded Lap Joint** design modules to support both tension and compression loading conditions. Previously, these modules were designed exclusively for tensile forces. Engineering applications often require joints to resist compressive loads, particularly in truss members, column splices, and bracing systems. The challenge was to modify the design logic, user interface, and verification procedures to seamlessly handle both force directions while maintaining full compliance with IS 800:2007. This required implementing different design checks for tension (Clause 6) versus compression (Clause 7), updating the input/output interface to accept axial forces, and adding a design mode selector to allow users to specify whether the connection is designed for tension or compression.

2.2 Task 1: Tasks Done

The integration of compression force support was a comprehensive task that required modifications across multiple layers of both welded connection modules. The work involved UI enhancements, core design logic modifications, and updated reporting capabilities.

2.2.1 Subtask 1.1: Welded Butt Joint Modifications

The **Butt Joint Welded** module (`butt_joint_welded.py`) was enhanced to support compression forces.

User Interface Updates (Butt Joint)

- **Design Tab:** Added a new "Design" preference tab with a "Design For" combobox (Tension/Compression).
- **Input Field:** Changed `KEY_TENSILE_FORCE` to `KEY_AXIAL_FORCE` to reflect the bi-directional nature of the load.
- **Output:** Added `KEY_OUT_DISP_DESIGN_FOR` to the results.

Design Logic Changes (Butt Joint)

The core change involved the `check_base_metal_strength` method. The logic now branches based on the design mode:

- **Compression:** Checks Gross Section Yielding (Cl. 7.1.2): $P_d = A_g \cdot f_y / \gamma_{m0}$.
- **Tension:** Checks minimum of Gross Yielding (Cl. 6.2) and Net Rupture (Cl. 6.3).

Listing 2.1: Butt Joint Base Metal Check for Compression

```
1 def check_base_metal_strength(self, design_dictionary):
2     """Check strength of base metal according to IS 800:2007.
3     Tension: yielding and rupture (Cl. 6). Compression: gross yielding
          (Cl. 7).
4     """
5     # ... (omitted setup) ...
```

```
    if self.design_for == 'Compression':
        # Compression: use gross area yielding (buckling is member-
            level, not joint)
        self.T_db = self.A_g * self.fy / self.gamma_m0
        base_metal_utilization = self.axial_force / self.T_db
    else:
        # Tension: yielding and rupture, take minimum
        T_dy = self.A_g * self.fy / self.gamma_m0
        T_du = 0.9 * self.A_n * self.fu / self.gamma_m1
        self.T_db = min(T_dy, T_du)
        base_metal_utilization = self.axial_force / self.T_db
```

2.2.2 Subtask 1.2: Welded Lap Joint Modifications

The **Lap Joint Welded** module (`lap_joint_welded.py`) underwent significant refactoring to integrate compression support and improve the overall design flow.

User Interface Updates (Lap Joint)

Similar to the butt joint, the lap joint module received the "Design" tab and "Axial Force" input updates.

Design Logic Changes (Lap Joint)

The `check_base_metal_strength` method was similarly updated but includes a shear lag factor specific to lap joints in tension.

Listing 2.2: Lap Joint Base Metal Check for Compression

```
def check_base_metal_strength(self, design_dictionary):
    logger.info(": ============== Base Metal Strength Check
        ==============")
    # IS800:2007 Cl.6.2.2, 6.2.3, 6.3 (shear lag), Cl.7.1.2 (
        compression)
    Tmin = min(float(design_dictionary[KEY_PLATE1_THICKNESS]), float(
        design_dictionary[KEY_PLATE2_THICKNESS]))
    self.A_g = Tmin * self.width
    self.gamma_m0 = 1.10
    self.gamma_m1 = 1.25
```

```
8
9       if self.design_for == 'Compression':
10          # Compression: use gross area yielding (Cl.7.1.2)
11          self.T_db = self.A_g * self.plate1.fy / self.gamma_m0
12          logger.info(f": Design strength of plate in compression = {self
                .T_db/1000:.2f} kN [Cl.7.1.2]")
13      else:
14          # Tension: yielding and rupture, take minimum (Cl.6.2.2, 6.2.3,
                6.3.3)
15          # Shear lag factor (Cl.6.3.3): For lap joints, net section
                efficiency = 0.7
16          shear_lag_factor = 0.7
17          T_dg = self.A_g * self.plate1.fy / self.gamma_m0  # Gross
                section yielding (Cl.6.2.2)
18          T_dn = 0.9 * self.A_g * self.plate1.fu * shear_lag_factor /
                self.gamma_m1  # Net section rupture (Cl.6.2.3, 6.3.3)
19          self.T_db = min(T_dg, T_dn)
20          logger.info(f": Design strength of plate in tension = {self.
                T_db/1000:.2f} kN [Cl.6.2.2, 6.2.3, 6.3.3]")
21
22      self.utilization_ratios['base_metal'] = self.axial_force / self.
            T_db if self.T_db > 0 else float('inf')
```

2.2.3 Summary of Impact

The integration ensures that both welded connection types can now safely handle compressive loads.

- **Code Compliance:** Full adherence to IS 800:2007 Clause 7 for compression members.
- **Safety:** Proper reporting of utilization ratios for both weld and base metal prevents unsafe designs.
- **Usability:** The interface changes are intuitive and explicitly indicate the design mode.

2.3 Task 1: Documentation

The changes are documented in the code via updated docstrings and comments referencing the specific IS 800 clauses (Cl. 7.1.2, Cl. 6.2, etc.). The generated design reports also reflect the design mode ("Compression" or "Tension") and use the corresponding formulas in the "Base Metal Capacity" check section.

2.3.1 Full code

The following command imports the entire source code for the module directly into the report for complete reference. This assumes the file `butt_joint_welded_diff.py` is placed in a `codes` sub-directory.

```
diff --git a/src/osdag/design_type/connection/butt_joint_welded.
   py b/src/osdag/design_type/connection/butt_joint_welded.py
index 1ac7507e..ddaab285 100644
--- a/src/osdag/design_type/connection/butt_joint_welded.py
+++ b/src/osdag/design_type/connection/butt_joint_welded.py
@@ -57,6 +57,7 @@ class ButtJointWelded(MomentConnection):
         tabs = []
         tabs.append((("Weld", TYPE_TAB_2, self.weld_values))) #
            added this line t.s.
         tabs.append(("Detailing", TYPE_TAB_2, self.
            detailing_values))
+        tabs.append(("Design", TYPE_TAB_2, self.design_values))
         return tabs

     def tab_value_changed(self):
@@ -76,6 +77,7 @@ class ButtJointWelded(MomentConnection):
             KEY_DP_DETAILING_EDGE_TYPE,
             KEY_DP_DETAILING_PACKING_PLATE
         ]))
+        design_input.append(("Design", TYPE_COMBOBOX, [
   KEY_DESIGN_FOR]))
         return design_input
```

```
     def input_dictionary_without_design_pref(self):
@@ -84,7 +86,8 @@ class ButtJointWelded(MomentConnection):
             KEY_DP_WELD_TYPE,
             KEY_DP_WELD_MATERIAL_G_O,
             KEY_DP_DETAILING_EDGE_TYPE,
-            KEY_DP_DETAILING_PACKING_PLATE
+            KEY_DP_DETAILING_PACKING_PLATE,
+            KEY_DESIGN_FOR
         ], ''))
         return design_input

@@ -100,10 +103,25 @@ class ButtJointWelded(MomentConnection):
             KEY_DP_WELD_TYPE: "Shop weld",
             KEY_DP_WELD_MATERIAL_G_O: str(fu),  # Set weld
                material grade to fu of selected material
             KEY_DP_DETAILING_EDGE_TYPE: "Sheared or hand flame
                cut",
-            KEY_DP_DETAILING_PACKING_PLATE: "Yes"
+            KEY_DP_DETAILING_PACKING_PLATE: "Yes",
+            KEY_DESIGN_FOR: "Tension"
         }
         return defaults.get(key)

+    def design_values(self, input_dictionary):
+        """Content of the 'Design' tab in Design Preferences."""
+        values = {
+            KEY_DESIGN_FOR: 'Tension',
+        }
+        if input_dictionary and KEY_DESIGN_FOR in
   input_dictionary:
+            values[KEY_DESIGN_FOR] = input_dictionary[
   KEY_DESIGN_FOR]
+
```

```
+         design_tab_content = []
+         t1 = (KEY_DESIGN_FOR, KEY_DISP_DESIGN_FOR, TYPE_COMBOBOX
    ,
+               ['Tension', 'Compression'], values[KEY_DESIGN_FOR
    ])
+         design_tab_content.append(t1)
+         return design_tab_content
+
      def detailing_values(self, input_dictionary):
          values = {
              KEY_DP_DETAILING_EDGE_TYPE: 'Sheared or hand flame
                 cut',
@@ -274,7 +292,7 @@ class ButtJointWelded(MomentConnection):
          t7 = (None, DISP_TITLE_FSL, TYPE_TITLE, None, True, 'No
             Validator')
          options_list.append(t7)

-         t17 = (KEY_TENSILE_FORCE, KEY_DISP_TENSILE_FORCE,
    TYPE_TEXTBOX, None, True, 'Int Validator')
+         t17 = (KEY_AXIAL_FORCE, KEY_DISP_AXIAL_FORCE,
    TYPE_TEXTBOX, None, True, 'Float Validator')
          options_list.append(t17)

          return options_list
@@ -373,6 +391,10 @@ class ButtJointWelded(MomentConnection):
                 round(self.weld_length_provided, 1) if flag else
                    '', True)
          out_list.append(t27)

+         t29 = (KEY_OUT_DESIGN_FOR, KEY_OUT_DISP_DESIGN_FOR,
    TYPE_TEXTBOX,
+                self.design_for if flag else '', True)
+         out_list.append(t29)
+
```

```
         return out_list

     def module_name(self):
@@ -402,7 +424,7 @@ class ButtJointWelded(MomentConnection):
         t37 = ('Plate2', self.call_3DPlate)
         components.append(t37)

-        return componentsconda
+        return components

     def call_3DPlate(self, ui, bgcolor):
         from PyQt5.QtWidgets import QCheckBox
@@ -444,10 +466,10 @@ class ButtJointWelded(MomentConnection):
                         else:
                             flag1 = True

-                    if option[2] == TYPE_TEXTBOX and option[0]
   == KEY_TENSILE_FORCE:
+                    if option[2] == TYPE_TEXTBOX and option[0]
   == KEY_AXIAL_FORCE:

                         if float(design_dictionary[option[0]])
                            <= 0.0:
-                            error = "Input value(s) cannot be
   equal or less than zero."
+                            error = "Input value for Axial Force
    must be a positive value."
                             all_errors.append(error)
                         else:
                             flag2 = True
@@ -486,7 +508,15 @@ class ButtJointWelded(MomentConnection):

         # self.plate_thickness =
            [3,4,6,8,10,12,14,16,20,22,24,25,26,28,30,32,36,40,45,50,56,63,8
```

```
        self.main_material = design_dictionary[KEY_MATERIAL]
-       self.tensile_force = float(design_dictionary[
    KEY_TENSILE_FORCE])*1000
+       # Design mode: default to Tension if not provided
+       self.design_for = design_dictionary.get(KEY_DESIGN_FOR,
    'Tension')
+       # Axial force: prefer KEY_AXIAL_FORCE, fallback to
    KEY_AXIAL, then KEY_TENSILE_FORCE
+       axial_kN_str = design_dictionary.get(KEY_AXIAL_FORCE,
+               design_dictionary.get(KEY_AXIAL,
+               design_dictionary.get(KEY_TENSILE_FORCE, 0)))
+       self.axial_force = abs(float(axial_kN_str)) * 1000  # N,
     always positive magnitude
+       # Maintain backward compatibility: many methods use
    tensile_force name
+       self.tensile_force = self.axial_force
        self.width = design_dictionary[KEY_PLATE_WIDTH]

        # print(self.sizelist)
@@ -791,7 +821,7 @@ class ButtJointWelded(MomentConnection):
        self.weld_strength = self.f_w * 0.707 * self.weld_size *
            self.weld_length_effective * self.N_f

        # Calculate weld utilization ratio
-       weld_utilization = self.tensile_force / self.
    weld_strength
+       weld_utilization = self.axial_force / self.weld_strength
        self.utilization_ratios['weld'] = weld_utilization

        #logger.info(": Weld Strength Calculation Results:")
@@ -835,7 +865,7 @@ class ButtJointWelded(MomentConnection):
        self.weld_strength_reduced = self.f_w_adjusted * 0.707 *
            self.weld_size * self.weld_length_effective * self.
```

```
            N_f

        # Update weld utilization with reduced strength
-       weld_utilization_reduced = self.tensile_force / self.
    weld_strength_reduced
+       weld_utilization_reduced = self.axial_force / self.
    weld_strength_reduced
        self.utilization_ratios['weld'] =
            weld_utilization_reduced  # Update the utilization
            ratio

        #logger.info(": Long joint reduction check results:")
@@ -845,7 +875,9 @@ class ButtJointWelded(MomentConnection):
        #logger.info(": Updated weld utilization ratio = {}".
            format(round(weld_utilization_reduced, 3)))

    def check_base_metal_strength(self, design_dictionary):
-       """Check strength of base metal according to IS 800:2007
    """
+       """Check strength of base metal according to IS
    800:2007.
+       Tension: yielding and rupture (Cl. 6). Compression:
    gross yielding (Cl. 7).
+       """

        # Extract material properties and handle material grade
            strings
        material_grade = design_dictionary[KEY_MATERIAL]
@@ -874,15 +906,16 @@ class ButtJointWelded(MomentConnection):
        self.A_g = Tmin * self.plates_width
        self.A_n = self.A_g  # For welded joints, net area
            equals gross area

-       # Calculate design strength based on yielding and
```

```
    rupture
-         T_dy = self.A_g * self.fy / self.gamma_m0
-         T_du = 0.9 * self.A_n * self.fu / self.gamma_m1
-
-         # Design base metal strength is minimum of the two
-         self.T_db = min(T_dy, T_du)
-
-         # Calculate base metal utilization ratio
-         base_metal_utilization = self.tensile_force/self.T_db
+         if self.design_for == 'Compression':
+             # Compression: use gross area yielding (buckling is
    member-level, not joint)
+             self.T_db = self.A_g * self.fy / self.gamma_m0
+             base_metal_utilization = self.axial_force / self.
    T_db
+         else:
+             # Tension: yielding and rupture, take minimum
+             T_dy = self.A_g * self.fy / self.gamma_m0
+             T_du = 0.9 * self.A_n * self.fu / self.gamma_m1
+             self.T_db = min(T_dy, T_du)
+             base_metal_utilization = self.axial_force / self.
    T_db
          self.utilization_ratios['base_metal'] =
             base_metal_utilization

         #logger.info(": Base Metal Strength Results:")
@@ -897,10 +930,15 @@ class ButtJointWelded(MomentConnection):
         #logger.info(": Base metal utilization ratio = {}".
             format(round(base_metal_utilization, 3)))

         if base_metal_utilization > 1:
-             logger.error(": Base metal strength is insufficient
    [cl. 6.2, IS 800:2007]")
-             logger.error(": Section fails in tension, try
```

```
    increasing the plate dimensions or using higher grade material
    ")
+            if self.design_for == 'Compression':
+                logger.error(": Base metal strength in
    compression is insufficient [cl. 7, IS 800:2007]")
+            else:
+                logger.error(": Base metal strength in tension
    is insufficient [cl. 6, IS 800:2007]")
         else:
-            logger.info(": Base metal strength is adequate")
+            if self.design_for == 'Compression':
+                logger.info(": Base metal strength in
    compression is adequate")
+            else:
+                logger.info(": Base metal strength in tension is
     adequate")

     def calculate_final_utilization_ratio(self):
         """Calculate final utilization ratio and set design
            status after all component checks"""
@@ -959,7 +997,8 @@ class ButtJointWelded(MomentConnection):
             self.report_input = {
                 KEY_MODULE: getattr(self, 'module', 'Butt Joint
                    Welded Connection'),
                 KEY_DISP_MATERIAL: safe_get('main_material', 'E
                    250 (Fe 410 W)A'),
-                KEY_DISP_TENSILE_FORCE: round(safe_get('
    tensile_force', 0)/1000, 2),
+                KEY_DISP_AXIAL: round(safe_get('axial_force', 0)
    /1000, 2),  # Changed from tensile_force
+                KEY_DISP_DESIGN_FOR: safe_get('design_for', '
    Tension'),
                 KEY_DISP_PLATE1_THICKNESS: safe_get('plate1.
                    thickness[0]', 8.0),
```

```
                KEY_DISP_PLATE2_THICKNESS: safe_get('plate2.
                    thickness[0]', 8.0),
                KEY_DISP_PLATE_WIDTH: safe_get('plates_width',
                    20.0),
@@ -1083,10 +1122,17 @@ class ButtJointWelded(MomentConnection):

                T_db = min(T_dy, T_du)

-               t1 = ('Base Metal Capacity',
-                   NoEscape(f'\\small T$_{{db}}$ = min(T$_{{dy
    }}$, T$_{{du}}$) = min({round(T_dy/1000, 2)}, {round(T_du
    /1000, 2)}) = {round(T_db/1000, 2)}~kN~[Ref:~IS~800:2007, Cl
    .6.2, 6.3]'),
-                   f'{round(T_db/1000, 2)} kN',
-                   'Pass')
+               # For base metal capacity section, conditionally
     format based on design_for
+               if safe_get('design_for') == 'Compression':
+                   t1 = ('Base Metal Capacity',
+                       NoEscape(f'\\small P$_{{d}}$ = A$_{{g}}$
    $\\times$ f$_{{y}}$/$\\gamma$$_{{m0}}$ = {A_g} $\\times$ {fy
    }/{gamma_m0} = {round(T_db/1000, 2)}~kN~[Ref:~IS~800:2007, Cl
    .7.1.2]'),
+                       f'{round(T_db/1000, 2)} kN',
+                       'Pass')
+               else:
+                   t1 = ('Base Metal Capacity',
+                       NoEscape(f'\\small T$_{{db}}$ = min(T$_
    {{dy}}$, T$_{{du}}$) = min({round(T_dy/1000, 2)}, {round(T_du
    /1000, 2)}) = {round(T_db/1000, 2)}~kN~[Ref:~IS~800:2007, Cl
    .6.2, 6.3]'),
+                       f'{round(T_db/1000, 2)} kN',
+                       'Pass')
                self.report_check.append(t1)
```

```
                # Detailing Requirements Section
@@ -1220,6 +1266,7 @@ class ButtJointWelded(MomentConnection):

             # Connection details
             KEY_DISP_AXIAL: round(self.tensile_force/1000, 2),
                 # Convert N to kN
+            KEY_DISP_DESIGN_FOR: self.design_for,

             # Connecting Members
             "Connecting Members": "TITLE",
```

The following command imports the entire source code for the module directly into the report for complete reference. This assumes the file `lap_joint_welded_diff.py` is placed in a `codes` sub-directory.

```
diff --git a/src/osdag/design_type/connection/lap_joint_welded.py
    b/src/osdag/design_type/connection/lap_joint_welded.py
index 222f6b76..e9dbb27e 100644
--- a/src/osdag/design_type/connection/lap_joint_welded.py
+++ b/src/osdag/design_type/connection/lap_joint_welded.py
@@ -1,13 +1,13 @@
 """
-Module: lap_joint_bolted.py
-Author: Aman
-Date: 2025-02-18
+Module: lap_joint_welded.py
+Author: Aman, Nishi Kant Mandal, Tanu Singh
+Date: 2025-07-14

 Description:
-    LapJointBolted is a moment connection module that represents
-    a bolted lap joint connection.
+    LapJointWelded is a moment connection module that represents
+    a welded lap joint connection.
```

```
     It inherits from MomentConnection and follows the same
         structure and design logic as other
     connection modules (e.g., BeamCoverPlate, ColumnCoverPlate)
         used in Osdag.
-
+
 Reference:
     - Osdag software guidelines and connection module structure
         documentation
 """
@@ -23,28 +23,40 @@ import logging

 import math

+from PyQt5.QtCore import Qt
+
 class LapJointWelded(MomentConnection):
     def __init__(self):
         super(LapJointWelded, self).__init__()
         self.design_status = False
-        self.spacing = None
+        self.weld_size = None
+        self.weld_length_provided = None
+        self.weld_strength = None
+        self.weld_thickness = None
+        self.plate_width = None
+        self.plate_length = None
+        self.plate_thickness = None
+        self.weld_type = None
+        self.weld_material = None
+        self.weld_fabrication = None
+        self.weld_angle = None
+        self.weld_length_effective = None
```

```
     ##################################################
     # Design Preference Functions Start
     ##################################################
     def tab_list(self):
         tabs = []
-        # Only Bolt and Detailing tabs
         tabs.append(("Weld", TYPE_TAB_2, self.weld_values))
         tabs.append(("Detailing", TYPE_TAB_2, self.
            detailing_values))
+        tabs.append(("Design", TYPE_TAB_2, self.design_values))
    # Add design tab
         return tabs

     def tab_value_changed(self):
-        # No tab value dependencies needed for bolt and
   detailing
         return []

     def edit_tabs(self):
-        return []  # Keep original empty implementation
+        return []

     def input_dictionary_design_pref(self):
         design_input = []
@@ -55,6 +67,7 @@ class LapJointWelded(MomentConnection):
         design_input.append(("Detailing", TYPE_COMBOBOX, [
             KEY_DP_DETAILING_EDGE_TYPE,
         ]))
+        design_input.append(("Design", TYPE_COMBOBOX, [
   KEY_DESIGN_FOR]))  # Add design preference
         return design_input

     def input_dictionary_without_design_pref(self):
@@ -63,23 +76,36 @@ class LapJointWelded(MomentConnection):
```

```
            KEY_DP_WELD_TYPE,
            KEY_DP_WELD_MATERIAL_G_O,
            KEY_DP_DETAILING_EDGE_TYPE,
+           KEY_DESIGN_FOR  # Add design preference
        ], ''))
        return design_input

    def get_values_for_design_pref(self, key, design_dictionary)
        :
-       # Get fu value from selected material
        if design_dictionary[KEY_MATERIAL] != 'Select Material':
            fu = Material(design_dictionary[KEY_MATERIAL], 41).
                fu
        else:
            fu = ''

-       # Default values as per requirements
        defaults = {
            KEY_DP_WELD_TYPE: "Shop weld",
-           KEY_DP_WELD_MATERIAL_G_O: str(fu),  # Set weld
    material grade to fu of selected material
+           KEY_DP_WELD_MATERIAL_G_O: str(fu),
            KEY_DP_DETAILING_EDGE_TYPE: "Sheared or hand flame
                cut",
        }
        return defaults.get(key)
+
+   def design_values(self, input_dictionary):
+       """Content of the 'Design' tab in Design Preferences."""
+       values = {
+                KEY_DESIGN_FOR: 'Tension',
+       }
+       if input_dictionary and KEY_DESIGN_FOR in
    input_dictionary:
```

```
+                values[KEY_DESIGN_FOR] = input_dictionary[
   KEY_DESIGN_FOR]
+
+        design_tab_content = []
+        t1 = (KEY_DESIGN_FOR, KEY_DISP_DESIGN_FOR, TYPE_COMBOBOX
   ,
+             ['Tension', 'Compression'], values[KEY_DESIGN_FOR])
+        design_tab_content.append(t1)
+        return design_tab_content

     def detailing_values(self, input_dictionary):
         values = {
@@ -91,8 +117,7 @@ class LapJointWelded(MomentConnection):
                 values[key] = input_dictionary[key]

         detailing = []
-
-        # Edge preparation method as per Cl. 10.2.4 of IS
   :800:2007
+
         t1 = (KEY_DP_DETAILING_EDGE_TYPE,
            KEY_DISP_DP_DETAILING_EDGE_TYPE, TYPE_COMBOBOX,
             ['Sheared or hand flame cut', 'Rolled, machine-flame
                 cut, sawn and planed'],
             values[KEY_DP_DETAILING_EDGE_TYPE])
@@ -103,41 +128,7 @@ class LapJointWelded(MomentConnection):

         return detailing

-    # def bolt_values(self, input_dictionary):
-    #     values = {
-    #         KEY_DP_BOLT_TYPE: 'Non Pre-tensioned',
-    #         KEY_DP_BOLT_HOLE_TYPE: 'Standard',
-    #         KEY_DP_BOLT_SLIP_FACTOR: '0.3'
```

```
-     #     }
-
-     #     for key in values.keys():
-     #         if key in input_dictionary.keys():
-     #             values[key] = input_dictionary[key]
-
-     #     bolt = []
-
-     #     # Bolt type selection
-     #     t1 = (KEY_DP_BOLT_TYPE, "Type", TYPE_COMBOBOX,
-     #         ['Non Pre-tensioned', 'Pre-tensioned'],
-     #         values[KEY_DP_BOLT_TYPE])
-     #     bolt.append(t1)
-
-     #     # Bolt hole type
-     #     t2 = (KEY_DP_BOLT_HOLE_TYPE, "Bolt Hole",
    TYPE_COMBOBOX,
-     #         ['Standard', 'Over-sized'],
-     #         values[KEY_DP_BOLT_HOLE_TYPE])
-     #     bolt.append(t2)
-
-     #     # Slip factor as per Table 20 of IS 800
-     #     t3 = (KEY_DP_BOLT_SLIP_FACTOR, "Slip Factor",
    TYPE_COMBOBOX,
-     #         ['0.3', '0.45', '0.5'],
-     #         values[KEY_DP_BOLT_SLIP_FACTOR])
-     #     bolt.append(t3)
-
-     #     return bolt
-
      def weld_values(self, input_dictionary):
-         # Get fu value from selected material if available
          fu = ''
          if input_dictionary and KEY_MATERIAL in input_dictionary
```

```
            :
                if input_dictionary[KEY_MATERIAL] != 'Select
                    Material':
@@ -145,10 +136,9 @@ class LapJointWelded(MomentConnection):

         values = {
             KEY_DP_WELD_TYPE: 'Shop weld',
-            KEY_DP_WELD_MATERIAL_G_O: str(fu) if fu else '410',
     # Default to 410 if no material selected
+            KEY_DP_WELD_MATERIAL_G_O: str(fu) if fu else '410',
         }

-        # Update values from input dictionary if available
         for key in values.keys():
             if input_dictionary and key in input_dictionary:
                 values[key] = input_dictionary[key]
@@ -156,43 +146,28 @@ class LapJointWelded(MomentConnection):
         weld = []

         t3 = (KEY_DP_WELD_TYPE, "Type", TYPE_COMBOBOX,
-            ['Shop weld', 'Field weld'],
+            ['Shop weld', 'Field weld'],
             values[KEY_DP_WELD_TYPE])
         weld.append(t3)

         t2 = (KEY_DP_WELD_MATERIAL_G_O, "Material Grade
             Overwrite, Fu (MPa)", TYPE_TEXTBOX,
-            None,
+            None,
             values[KEY_DP_WELD_MATERIAL_G_O])
         weld.append(t2)
         return weld

-
```

```
-     ####################################
-     # Design Preference Functions End
-     ####################################
-
      def set_osdaglogger(key):
-
-         """
-         Function to set Logger for Tension Module
-         """
-
-         # @author Arsil Zunzunia
          global logger
          logger = logging.getLogger('Osdag')
-
          logger.setLevel(logging.DEBUG)
          handler = logging.StreamHandler()
          formatter = logging.Formatter(fmt='%(asctime)s - %(name)
             s - %(levelname)s - %(message)s', datefmt='%Y-%m-%d %
             H:%M:%S')
-
          handler.setFormatter(formatter)
          logger.addHandler(handler)
          handler = logging.FileHandler('logging_text.log')
-
          formatter = logging.Formatter(fmt='%(asctime)s - %(name)
             s - %(levelname)s - %(message)s', datefmt='%Y-%m-%d %
             H:%M:%S')
          handler.setFormatter(formatter)
          logger.addHandler(handler)
-
          if key is not None:
              handler = OurLog(key)
              formatter = logging.Formatter(fmt='%(asctime)s - %(
                 name)s - %(levelname)s - %(message)s',
```

```
@@ -200,163 +175,127 @@ class LapJointWelded(MomentConnection):
             handler.setFormatter(formatter)
             logger.addHandler(handler)

-
     def input_value_changed(self):
-
         lst = []
-
         t8 = ([KEY_MATERIAL], KEY_MATERIAL, TYPE_CUSTOM_MATERIAL
            , self.new_material)
         lst.append(t8)
-
         return lst

-
     def input_values(self):
-
         options_list = []
-
-        t16 = (KEY_MODULE, KEY_DISP_LAPJOINTBOLTED, TYPE_MODULE,
    None, True, 'No Validator')
+        t16 = (KEY_MODULE, KEY_DISP_LAPJOINTWELDED, TYPE_MODULE,
    None, True, 'No Validator')
         options_list.append(t16)
-
         t1 = (None, DISP_TITLE_CM, TYPE_TITLE, None, True, 'No
            Validator')
         options_list.append(t1)
-
         t31 = (KEY_PLATE1_THICKNESS, KEY_DISP_PLATE1_THICKNESS,
            TYPE_COMBOBOX, VALUES_PLATETHK_CUSTOMIZED, True, 'Int
            Validator')
         options_list.append(t31)
```

```
-
        t34 = (KEY_PLATE2_THICKNESS, KEY_DISP_PLATE2_THICKNESS,
            TYPE_COMBOBOX, VALUES_PLATETHK_CUSTOMIZED, True, 'Int
             Validator')
        options_list.append(t34)
-
        t35 = (KEY_PLATE_WIDTH, KEY_DISP_PLATE_WIDTH,
            TYPE_TEXTBOX, None, True, 'Float Validator')
        options_list.append(t35)
-
        t5 = (KEY_MATERIAL, KEY_DISP_MATERIAL, TYPE_COMBOBOX,
            VALUES_MATERIAL, True, 'No Validator')
        options_list.append(t5)
-
        t18 = (None, DISP_TITLE_WELD, TYPE_TITLE, None, True, '
            No Validator')
        options_list.append(t18)
-
        t20 = (KEY_WELD_SIZE, KEY_DISP_WELD_SIZE,
            TYPE_COMBOBOX_CUSTOMIZED, VALUES_ALL_CUSTOMIZED, True
            , 'No Validator')
        options_list.append(t20)
-
        t6 = (None, DISP_TITLE_FSL, TYPE_TITLE, None, True, 'No
            Validator')
        options_list.append(t6)
-
-       t17 = (KEY_TENSILE_FORCE, KEY_DISP_TENSILE_FORCE,
   TYPE_TEXTBOX, None, True, 'Int Validator')
+       # Replace tensile force with axial force
+       t17 = (KEY_AXIAL_FORCE, KEY_DISP_AXIAL_FORCE,
   TYPE_TEXTBOX, None, True, 'Int Validator')
        options_list.append(t17)
-
```

```
        return options_list

    def customized_input(self):
-
        list1 = []
        t11 = (KEY_WELD_SIZE, self.weld_size_customized)
        list1.append(t11)
        return list1
-
+
    @staticmethod
    def weld_size_customized():
        return [str(size) for size in WELD_SIZES]
-
-
-    def spacing(self, status):
-        spacing = []
-
-        t00 = (None, "", TYPE_NOTE, "Representative Image for
  Spacing Details - 3 x 3 pattern considered")
-        spacing.append(t00)
-
-        t99 = (None, 'Spacing Details', TYPE_SECTION,
-            [str(files("osdag.data.ResourceFiles.images").
  joinpath("spacing_3.png")), 400, 277, ""])  # [image, width,
  height, caption]
-        spacing.append(t99)
-
-        t9 = (KEY_OUT_PITCH, KEY_OUT_DISP_PITCH, TYPE_TEXTBOX,
  self.plate.gauge_provided if status else '')
-        spacing.append(t9)
-
-        t10 = (KEY_OUT_END_DIST, KEY_OUT_DISP_END_DIST,
  TYPE_TEXTBOX, self.plate.edge_dist_provided if status else '')
```

```
-         spacing.append(t10)
-
-         t11 = (KEY_OUT_GAUGE, KEY_OUT_DISP_GAUGE, TYPE_TEXTBOX,
    self.plate.pitch_provided if status else '')
-         spacing.append(t11)
-
-         t12 = (KEY_OUT_EDGE_DIST, KEY_OUT_DISP_EDGE_DIST,
    TYPE_TEXTBOX, self.plate.end_dist_provided if status else '')
-         spacing.append(t12)
-
-         return spacing

      def output_values(self, flag):
-         out_list=[]
-         # Cover plate details
-         # Calculate cover_type based on planes attribute
+         out_list = []
          t21 = (None, DISP_TITLE_WELD, TYPE_TITLE, None, True)
          out_list.append(t21)
-
          t22 = (KEY_OUT_UTILISATION_RATIO,
             KEY_OUT_DISP_UTILISATION_RATIO, TYPE_TEXTBOX,
-                round(self.utilization_ratio, 3) if flag else '',
     True)
+                round(self.utilization_ratio, 3) if flag and
    hasattr(self, 'utilization_ratio') and self.utilization_ratio
    is not None else '', True)
          out_list.append(t22)
-
          t23 = (KEY_OUT_WELD_TYPE, KEY_OUT_DISP_WELD_TYPE,
             TYPE_TEXTBOX,
                 "Fillet" if flag else '', True)
          out_list.append(t23)
-
```

```
        t24 = (KEY_OUT_WELD_SIZE, KEY_OUT_DISP_WELD_SIZE,
            TYPE_TEXTBOX,
-               round(self.weld_size, 1) if flag else '', True)
+               round(self.weld_size, 1) if flag and self.
    weld_size is not None else '', True)
        out_list.append(t24)
-
        t25 = (KEY_OUT_WELD_STRENGTH,
            KEY_OUT_DISP_WELD_STRENGTH_kN, TYPE_TEXTBOX,
-               round(self.weld_strength/1000, 2) if flag else ''
    , True)  # Convert to kN
+               round(self.weld_strength / 1000, 2) if flag and
    hasattr(self, 'weld_strength') and self.weld_strength is not
    None else '', True)
        out_list.append(t25)
-
        t26 = (KEY_OUT_WELD_LENGTH_EFF,
            KEY_OUT_DISP_WELD_LENGTH_EFF, TYPE_TEXTBOX,
-               round(self.weld_length_effective, 1) if flag else
     '', True)
+               round(self.weld_length_effective, 1) if flag and
    self.weld_length_effective is not None else '', True)
        out_list.append(t26)
-
-       t27 = (KEY_OUT_BOLT_CONN_LEN, KEY_OUT_DISP_BOLT_CONN_LEN
    , TYPE_TEXTBOX,
-               round(self.weld_length_provided, 1) if flag else
    '', True)
+       t27 = (KEY_OUT_WELD_CONN_LEN, KEY_OUT_DISP_WELD_CONN_LEN
    , TYPE_TEXTBOX,
+               round(self.connection_length, 1) if flag and
    hasattr(self, 'connection_length') and self.connection_length
    is not None else '', True)
        out_list.append(t27)
```

```
-
+        t29 = (KEY_OUT_DESIGN_FOR, KEY_OUT_DISP_DESIGN_FOR,
   TYPE_TEXTBOX,
+               self.design_for if flag and hasattr(self, '
   design_for') else '', True)
+        out_list.append(t29)
         return out_list

     def module_name(self):
+        return KEY_DISP_LAPJOINTWELDED

-        return KEY_DISP_LAPJOINTBOLTED
-
-    def func_for_validation(self, design_dictionary):
+    def call_3DColumn(self, ui, bgcolor):
+        if ui.chkBxCol.isChecked():
+            ui.btn3D.setChecked(Qt.Unchecked)
+            ui.chkBxCol.setChecked(Qt.Unchecked)
+            ui.mytabWidget.setCurrentIndex(0)
+        ui.commLogicObj.display_3DModel("Column", bgcolor)

+    def get_3d_components(self):
+        components = []
+        t1 = ('Model', self.call_3DModel)
+        components.append(t1)
+        t3 = ('Plate1', self.call_3DColumn)
+        components.append(t3)
+        t4 = ('Plate2', self.call_3DPlate)
+        components.append(t4)
+        return components
+
+    def call_3DPlate(self, ui, bgcolor):
+        from PyQt5.QtWidgets import QCheckBox
+        from PyQt5.QtCore import Qt
```

```
+         for chkbox in ui.frame.children():
+             if chkbox.objectName() == 'Cover Plate':
+                 continue
+             if isinstance(chkbox, QCheckBox):
+                 chkbox.setChecked(Qt.Unchecked)
+         ui.commLogicObj.display_3DModel("Cover Plate", bgcolor)
+
+     def func_for_validation(self, design_dictionary):
          all_errors = []
-         "check valid inputs and empty inputs in input dock"
-         # print(design_dictionary,'
    djsgggggggggggggggggggggggggggggggggggggggggggggggggggggg')
          self.design_status = False
-
          flag = False
          flag1 = False
          flag2 = False
-         # self.include_status = True

          option_list = self.input_values(self)
          missing_fields_list = []

-         print(f'\n func_for_validation option list = {
    option_list}'
-               f'\n  design_dictionary {design_dictionary}')
-
          for option in option_list:
              if option[2] == TYPE_TEXTBOX:
                  if design_dictionary[option[0]] == '':
-
-                     print(f"\n option {option}")
-
                      missing_fields_list.append(option[1])
                  else:
```

```
-                     if option[2] == TYPE_TEXTBOX and option[0]
    == KEY_PLATE_WIDTH:
-                         # val = option[4]
-                         # print(design_dictionary[option[0]], "
    jhvhj")
+                     if option[0] == KEY_PLATE_WIDTH:
                          if float(design_dictionary[option[0]])
                             <= 0.0:
                              error = "Input value(s) cannot be
                                 equal or less than zero."
                              all_errors.append(error)
                          else:
                              flag1 = True
-
-                     if option[2] == TYPE_TEXTBOX and option[0]
    == KEY_TENSILE_FORCE:
-
+                     # Change from KEY_TENSILE_FORCE to
    KEY_AXIAL_FORCE
+                     elif option[0] == KEY_AXIAL_FORCE:
                          if float(design_dictionary[option[0]])
                             <= 0.0:
                              error = "Input value(s) cannot be
                                 equal or less than zero."
                              all_errors.append(error)
@@ -365,74 +304,307 @@ class LapJointWelded(MomentConnection):
                 else:
                     pass

-
         if len(missing_fields_list) > 0:
             error = self.generate_missing_fields_error_string(
                self, missing_fields_list)
             all_errors.append(error)
```

```
-             # flag = False
          else:
              flag = True
-         if flag  and flag1 and flag2:
-             # self.set_input_values(self, design_dictionary)
-             # print("DESIGN DICT" + str(design_dictionary))
-             print("succsess")
+
+         if flag and flag1 and flag2:
+             self.set_input_values(self, design_dictionary)
          else:
              return all_errors

+     def set_input_values(self, design_dictionary):
+         design_dictionary_with_defaults = design_dictionary.copy
    ()
+         if KEY_SHEAR not in design_dictionary_with_defaults:
+             design_dictionary_with_defaults[KEY_SHEAR] = 0.0
+         if KEY_AXIAL not in design_dictionary_with_defaults:
+             design_dictionary_with_defaults[KEY_AXIAL] = 0.0
+         if KEY_MOMENT not in design_dictionary_with_defaults:
+             design_dictionary_with_defaults[KEY_MOMENT] = 0.0
+
+         super(LapJointWelded, self).set_input_values(self,
    design_dictionary_with_defaults)
+         self.module = design_dictionary[KEY_MODULE]
+         self.mainmodule = "Lap Joint Welded Connection"
+         self.main_material = design_dictionary[KEY_MATERIAL]
+
+         # Design mode: default to Tension if not provided
+         self.design_for = design_dictionary.get(KEY_DESIGN_FOR,
    'Tension')
+
+         # Use axial force instead of tensile force
```

```
+        axial_kN_str = design_dictionary.get(KEY_AXIAL_FORCE,
+                design_dictionary.get(KEY_AXIAL,
+                design_dictionary.get(KEY_TENSILE_FORCE, 0)))
+
+        self.axial_force = abs(float(axial_kN_str)) * 1000  # N,
    always positive magnitude
+        # Maintain backward compatibility: many methods use
   tensile_force name
+        self.tensile_force = self.axial_force
+
+        self.width = float(design_dictionary[KEY_PLATE_WIDTH])
+        self.plate1 = Plate(thickness=[design_dictionary[
   KEY_PLATE1_THICKNESS]],
+                            material_grade=design_dictionary[
   KEY_MATERIAL],
+                            width=design_dictionary[KEY_PLATE_WIDTH
   ])
+        self.plate2 = Plate(thickness=[design_dictionary[
   KEY_PLATE2_THICKNESS]],
+                                material_grade=design_dictionary[
   KEY_MATERIAL],
+                                width=design_dictionary[
   KEY_PLATE_WIDTH])
+        self.weld = Weld(material_g_o=design_dictionary[
   KEY_DP_WELD_MATERIAL_G_O],
+                             type=design_dictionary[KEY_DP_WELD_TYPE
   ],
+                             fabrication=design_dictionary.get(
   KEY_DP_FAB_SHOP, KEY_DP_FAB_SHOP))
+        self.weld.size = design_dictionary[KEY_WELD_SIZE]
+        self.design_of_weld(self, design_dictionary)
+
+    def design_of_weld(self, design_dictionary):
+        logger.info(": ========== Design of Lap Joint Welded
```

```
   Connection ==========")
+        logger.info(": Design Approach: IS 800:2007 Clause 10.5"
   )
+        self.utilization_ratios = {}
+
+        if not self.weld_size_check(self, design_dictionary):
+            return
+
+        self.calculate_weld_strength(self, design_dictionary)
+        self.calculate_weld_length(self)
+        self.check_long_joint(self)
+        self.check_base_metal_strength(self, design_dictionary)
+        self.calculate_final_utilization_ratio(self)
+
+    def weld_size_check(self, design_dictionary):
+        logger.info(": ================ Weld Size Check
   ================")
+        weld_size = design_dictionary[KEY_WELD_SIZE]
+        plate1_thk = float(design_dictionary[
   KEY_PLATE1_THICKNESS])
+        plate2_thk = float(design_dictionary[
   KEY_PLATE2_THICKNESS])
+        Tmin = min(plate1_thk, plate2_thk)
+        s_min = IS800_2007.cl_10_5_2_3_min_weld_size(plate1_thk,
    plate2_thk)
+        s_max = Tmin - 1.5 if Tmin >= 10 else Tmin
+
+        logger.info(f": Minimum weld size required (s_min) = {
   s_min} mm [Ref. Table 21, Cl.10.5.2.3]")
+        logger.info(f": Maximum allowed weld size (s_max) = {
   s_max} mm [Ref. Cl.10.5.3.1]")
+
+        selected_size = None
+        if isinstance(weld_size, str) and weld_size.lower() == '
```

```
   all':
+            valid_sizes = [s for s in ALL_WELD_SIZES if s_min <=
    s <= s_max]
+            if valid_sizes:
+                selected_size = float(valid_sizes[0])
+        else:
+            try:
+                size_val = float(weld_size[0] if isinstance(
   weld_size, list) else weld_size)
+                if s_min <= size_val <= s_max:
+                    selected_size = size_val
+            except (ValueError, IndexError):
+                pass

-    def call_3DColumn(self, ui, bgcolor):
-        # status = self.resultObj['Bolt']['status']
-        # if status is True:
-        #     self.ui.chkBx_beamSec1.setChecked(Qt.Checked)
-        if ui.chkBxCol.isChecked():
-            ui.btn3D.setChecked(Qt.Unchecked)
-            ui.chkBxCol.setChecked(Qt.Unchecked)
-            ui.mytabWidget.setCurrentIndex(0)
-        # self.display_3DModel("Beam", bgcolor)
-        ui.commLogicObj.display_3DModel("Column", bgcolor)
-
-    def get_3d_components(self):
-        components = []
-
-        t1 = ('Model', self.call_3DModel)
-        components.append(t1)
-
-        t3 = ('Plate1', self.call_3DColumn)
-        components.append(t3)
-
```

```
-         t4 = ('Plate2', self.call_3DPlate)
-         components.append(t4)
-
-         return components
-
-     def call_3DPlate(self, ui, bgcolor):
-         from PyQt5.QtWidgets import QCheckBox
-         from PyQt5.QtCore import Qt
-         for chkbox in ui.frame.children():
-             if chkbox.objectName() == 'Cover Plate':
-                 continue
-             if isinstance(chkbox, QCheckBox):
-                 chkbox.setChecked(Qt.Unchecked)
-         ui.commLogicObj.display_3DModel("Cover Plate", bgcolor)
-
-     def warn_text(self):
-
-         """
-         Function to give logger warning when any old value is
    selected from Column and Beams table.
-         """
-
-         # @author Arsil Zunzunia
-         global logger
-         red_list = red_list_function()
-         if self.supported_section.designation in red_list or
    self.supporting_section.designation in red_list:
-             logger.warning(
-                 " : You are using a section (in red color) that
    is not available in latest version of IS 808")
-             logger.info(
-                 " : You are using a section (in red color) that
    is not available in latest version of IS 808")
-
```

```
-
-     ################################ Outlist Dict
    ##########################################################################
-
-
-
-     ################################ Design Report
    ##########################################################################

\ No newline at end of file
+         if selected_size is None:
+             logger.error(": Selected weld size is not suitable."
    )
+             self.design_status = False
+             return False
+
+         self.weld_size = selected_size
+         logger.info(f": Selected weld size = {self.weld_size} mm
     (Pass)")
+         return True
+
+     def calculate_weld_strength(self, design_dictionary):
+         logger.info(": ============== Weld Strength Calculation
    ==============")
+         # IS800:2007 Cl.10.5.3.2: Throat thickness a = K * s,
    where K depends on angle
+         # For fillet welds, K = sin(  ),     = weld angle (
    default 45   if not specified)
+         weld_angle = design_dictionary.get('weld_angle', 45)
+         if not isinstance(weld_angle, (int, float)):
+             try:
+                 weld_angle = float(weld_angle)
+             except Exception:
```

```
+                 weld_angle = 45
+         # K = sin(angle)
+         K = round(math.sin(math.radians(weld_angle)), 3)
+         if K <= 0:
+             logger.error(f": Invalid weld angle {weld_angle}  .
   Using default K=0.7 (45  )")
+             K = 0.7
+         self.effective_throat_thickness = K * self.weld_size  #
   Cl.10.5.3.2
+         logger.info(f": Effective throat thickness (a) = {self.
   effective_throat_thickness:.2f} mm [Cl.10.5.3.2, K={K},   ={
   weld_angle}  ]")
+         self.fu = float(design_dictionary[
   KEY_DP_WELD_MATERIAL_G_O])
+         self.gamma_mw = 1.25 if design_dictionary[
   KEY_DP_WELD_TYPE] == "Shop weld" else 1.50  # Cl.10.5.7.1
+         self.weld_design_strength = (self.fu * self.
   effective_throat_thickness) / (math.sqrt(3) * self.gamma_mw)
   # Cl.10.5.7.1
+         self.parent_design_strength = 0.6 * self.fu * self.
   effective_throat_thickness / self.gamma_mw  # Cl.10.5.7.2
+         self.fillet_weld_design_strength = min(self.
   weld_design_strength, self.parent_design_strength)
+         logger.info(f": Design strength of fillet weld = {self.
   fillet_weld_design_strength:.2f} N/mm^2 [Cl.10.5.7]")
+         # Weld stress check (Cl.10.5.7):
+         self.weld_stress = self.tensile_force / (2 * self.
   effective_throat_thickness * self.l_eff) if hasattr(self, '
   l_eff') and self.l_eff else 0
+         if self.weld_stress > self.fillet_weld_design_strength:
+             error_msg = f"Weld stress {self.weld_stress:.2f} N/
   mm^2 exceeds design strength {self.fillet_weld_design_strength
   :.2f} N/mm^2 [Cl.10.5.7]"
+             logger.error(": " + error_msg)
```

```
+            print("[Osdag ERROR]", error_msg)
+            self.design_status = False
+            self.design_error = "Weld stress exceeds design strength."
+            return
+
+    def calculate_weld_length(self):
+        logger.info(": ============== Weld Length Calculation ==============")
+        # Required effective weld length (Cl.10.5.4.1)
+        self.weld_length_required = self.tensile_force / (2 * self.fillet_weld_design_strength)
+        self.leff_min = max(4 * self.weld_size, 40)  # Cl.10.5.4.1
+        self.leff_max = 70 * self.weld_size  # Cl.10.5.4.1
+        logger.info(f": Required effective weld length = {self.weld_length_required:.2f} mm")
+        logger.info(f": Minimum effective weld length = {self.leff_min} mm [Cl.10.5.4.1]")
+        logger.info(f": Maximum effective weld length = {self.leff_max} mm [Cl.10.5.4.1]")
+        # Check min/max
+        if self.weld_length_required < self.leff_min:
+            self.l_eff = self.leff_min
+            logger.warning(f": Required length is less than minimum, using l_eff = {self.l_eff} mm [Cl.10.5.4.1]")
+        elif self.weld_length_required > self.leff_max:
+            logger.error(": Required weld length exceeds maximum allowed. Increase weld size. [Cl.10.5.4.1]")
+            self.design_status = False
+            raise ValueError("Required weld length exceeds maximum allowed.")
+        else:
+            self.l_eff = self.weld_length_required
```

```
+            logger.info(": Required weld length is within limits (Pass)")
+        # Detailing: Minimum spacing between parallel fillet welds (Cl.10.5.4.2)
+        # Not implemented here, but should be checked in GUI or input validation
+
+    def check_long_joint(self):
+        logger.info(": ============== Long Joint Check ==============")
+        # IS800:2007 Cl.10.5.7.3: Long joint reduction factor
+        self.beta_lw = 1.0
+        if self.l_eff > 150 * self.effective_throat_thickness:
+            self.beta_lw = 1.2 - 0.2 * (self.l_eff / (150 * self.effective_throat_thickness))
+            self.beta_lw = max(0.6, min(self.beta_lw, 1.0))
+            logger.info(f": Joint is long, reduction factor beta_lw = {self.beta_lw:.3f} [Cl.10.5.7.3]")
+        else:
+            logger.info(": No reduction for long joint required (Pass)")
+        # Modified required length
+        l_req_modified = self.l_eff / self.beta_lw
+        if l_req_modified < self.leff_min:
+            logger.warning(f": Modified required weld length {l_req_modified:.2f} mm is less than minimum effective length {self.leff_min} mm [Cl.10.5.4.1]")
+            self.l_eff = self.leff_min
+        elif l_req_modified > self.leff_max:
+            logger.error(": Modified required weld length exceeds maximum allowed. Increase weld size. [Cl.10.5.4.1]")
+            self.design_status = False
+            raise ValueError("Modified required weld length exceeds maximum allowed.")
```

```
+         else:
+             self.l_eff = l_req_modified
+         # End return length (Cl.10.5.4.5): min(2*s, 12mm)
+         self.end_return_length = max(2 * self.weld_size, 12)  # Cl.10.5.4.5
+         logger.info(f": End return length = {self.end_return_length} mm [Cl.10.5.4.5]")
+         # Overlap length (Cl.10.5.4.3): min overlap = 4*s or 40 mm, whichever is more
+         self.overlap_length = max(4 * self.weld_size, 40)
+         logger.info(f": Overlap length = {self.overlap_length} mm [Cl.10.5.4.3]")
+         self.connection_length = self.l_eff + 2 * self.end_return_length
+         # Design capacity (Cl.10.5.7.3):
+         self.design_capacity = 2 * self.l_eff * self.fillet_weld_design_strength * self.beta_lw
+         # Weld stress check (Cl.10.5.7):
+         self.weld_stress = self.tensile_force / (2 * self.effective_throat_thickness * self.l_eff) if self.l_eff else 0
+         if self.weld_stress > self.fillet_weld_design_strength:
+             error_msg = f"Weld stress {self.weld_stress:.2f} N/mm^2 exceeds design strength {self.fillet_weld_design_strength:.2f} N/mm^2 [Cl.10.5.7]"
+             logger.error(": " + error_msg)
+             print("[Osdag ERROR]", error_msg)
+             self.design_status = False
+             self.design_error = "Weld stress exceeds design strength."
+             return
+         self.utilization_ratios['weld'] = self.tensile_force / self.design_capacity if self.design_capacity > 0 else float('inf')
+         logger.info(f": Provided effective length = {self.l_eff
```

```
:.2f} mm")
+        logger.info(f": Design capacity of weld = {self.
design_capacity/1000:.2f} kN")
+
+    def check_base_metal_strength(self, design_dictionary):
+        logger.info(": ============== Base Metal Strength Check
==============")
+        # IS800:2007 Cl.6.2.2, 6.2.3, 6.3 (shear lag), Cl.7.1.2
(compression)
+        Tmin = min(float(design_dictionary[KEY_PLATE1_THICKNESS
]), float(design_dictionary[KEY_PLATE2_THICKNESS]))
+        self.A_g = Tmin * self.width
+        self.gamma_m0 = 1.10
+        self.gamma_m1 = 1.25
+
+        if self.design_for == 'Compression':
+            # Compression: use gross area yielding (Cl.7.1.2)
+            self.T_db = self.A_g * self.plate1.fy / self.
gamma_m0
+            logger.info(f": Design strength of plate in
compression = {self.T_db/1000:.2f} kN [Cl.7.1.2]")
+        else:
+            # Tension: yielding and rupture, take minimum (Cl
.6.2.2, 6.2.3, 6.3.3)
+            # Shear lag factor (Cl.6.3.3): For lap joints, net
section efficiency = 0.7
+            shear_lag_factor = 0.7
+            T_dg = self.A_g * self.plate1.fy / self.gamma_m0  #
Gross section yielding (Cl.6.2.2)
+            T_dn = 0.9 * self.A_g * self.plate1.fu *
shear_lag_factor / self.gamma_m1  # Net section rupture (Cl
.6.2.3, 6.3.3)
+            self.T_db = min(T_dg, T_dn)
+            logger.info(f": Design strength of plate in tension
```

```
   = {self.T_db/1000:.2f} kN [Cl.6.2.2, 6.2.3, 6.3.3]")
+
+        self.utilization_ratios['base_metal'] = self.axial_force
    / self.T_db if self.T_db > 0 else float('inf')
+
+    def calculate_final_utilization_ratio(self):
+        logger.info(": ============== Final Check ==============
   ")
+        # Eccentricity check (IS800:2007 Cl.10.5.7.4):
+        # For lap joints, if eccentricity exists, reduce design
   strength accordingly (not implemented, placeholder)
+        # TODO: Implement eccentricity reduction if required
+        self.utilization_ratio = max(self.utilization_ratios.
   values())
+        logger.info(f": Weld utilization ratio = {self.
   utilization_ratios['weld']:.3f}")
+        logger.info(f": Base metal utilization ratio = {self.
   utilization_ratios['base_metal']:.3f}")
+        logger.info(f": Overall utilization ratio = {self.
   utilization_ratio:.3f}")
+        if self.utilization_ratio > 1.0:
+            error_msg = "Design is UNSAFE. Utilization ratio
   exceeds 1.0."
+            logger.error(": " + error_msg)
+            print("[Osdag ERROR]", error_msg)
+            self.design_status = False
+            self.design_error = "Utilization ratio exceeds 1.0.
   Design is unsafe."
+            return
+        else:
+            logger.info(": Design is SAFE.")
+            self.design_status = True
+        self.weld_strength = self.design_capacity
+        self.weld_length_effective = self.l_eff
```

```
+
+    def save_design(self, popup_summary):
+        self.report_input = {
+            KEY_MODULE: self.module,
+            KEY_MAIN_MODULE: self.mainmodule,
+            KEY_DISP_AXIAL: round(self.axial_force / 1000, 2),
+            KEY_DISP_DESIGN_FOR: self.design_for,
+            KEY_DISP_PLATETHK: str([int(d) for d in [self.plate1
    .thickness[0], self.plate2.thickness[0]]]),
+            KEY_DISP_MATERIAL: self.main_material,
+            KEY_DISP_ULTIMATE_STRENGTH_REPORT: self.plate1.fu,
+            KEY_DISP_YIELD_STRENGTH_REPORT: self.plate1.fy,
+            KEY_DISP_PLATE_WIDTH: self.width,
+            "Weld Details - Input and Design Preference": "TITLE
    ",
+            KEY_DISP_DP_WELD_TYPE: self.weld.type,
+            KEY_DISP_DP_WELD_FAB: self.weld.fabrication,
+            KEY_DISP_DP_WELD_MATERIAL_G_O_REPORT: self.weld.fu,
+            KEY_DISP_WELD_SIZE: self.weld_size,
+            "Safety Factors": "TITLE",
+            KEY_DISP_GAMMA_MW: self.gamma_mw,
+            "Weld Angle (deg)": getattr(self, 'weld_angle', 45),
+            "Effective Throat Thickness (mm)": getattr(self, '
    effective_throat_thickness', None),
+            "Long Joint Reduction Factor  _lw ": getattr(self, '
    beta_lw', 1.0),
+            "End Return Length (mm)": getattr(self, '
    end_return_length', None),
+            "Overlap Length (mm)": getattr(self, 'overlap_length
    ', None),
+            "Shear Lag Factor": 0.7,
+        }
+
+        self.report_check = []
```

```
+        if self.design_status:
+            t1 = ('SubSection', 'Weld Design', '|p{3cm}|p{6.5cm}|p{5cm}|p{1cm}|')
+            self.report_check.append(t1)
+
+            # Add appropriate design capacity calculation based on design mode
+            if self.design_for == 'Compression':
+                t1 = ('Base Metal Capacity',
+                      f"P_d = A_g    f_y/ _m0   = {self.A_g:.2f}    {self.plate1.fy}/{self.gamma_m0} = {self.T_db/1000:.2f} kN [Ref: IS 800:2007, Cl.7.1.2]",
+                      f"{self.T_db/1000:.2f} kN",
+                      "Pass" if self.T_db >= self.axial_force else "Fail")
+            else:
+                t1 = ('Base Metal Capacity',
+                      f"min(T_dg, T_dn) = min({self.A_g * self.plate1.fy / self.gamma_m0/1000:.2f}, {0.9 * self.A_g * self.plate1.fu * 0.7 / self.gamma_m1/1000:.2f}) = {self.T_db/1000:.2f} kN [Ref: IS 800:2007, Cl.6.2, 6.3]",
+                      f"{self.T_db/1000:.2f} kN",
+                      "Pass" if self.T_db >= self.axial_force else "Fail")
+            self.report_check.append(t1)
+
+            t1 = (DISP_WELD_STRENGTH,
+                  f"Required: {self.tensile_force / 1000:.2f} kN",
+                  f"Provided: {self.design_capacity / 1000:.2f} kN",
+                  "Pass" if self.design_capacity >= self.tensile_force else "Fail")
+            self.report_check.append(t1)
```

```
+            t1 = ('Overall Utilization Ratio',
+                  "<= 1.0",
+                  f"{self.utilization_ratio:.3f}",
+                  "Pass" if self.utilization_ratio <= 1.0 else "
   Fail")
+            self.report_check.append(t1)
+        else:
+            t1 = ('SubSection', 'Design Status', '|p{3.5cm}|p
   {4.5cm}|p{6cm}|p{1.5cm}|')
+            self.report_check.append(t1)
+            t1 = ('Design Status', '', 'Design Fails', 'Fail')
+            self.report_check.append(t1)
+        fname_no_ext = popup_summary['filename']
+        CreateLatex.save_latex(CreateLatex(), self.report_input,
    self.report_check, popup_summary,
+                               fname_no_ext, os.path.abspath(".").
   replace("\\", "/"), [], "/ResourceFiles/images/3d.png", module
   =self.module)
\ No newline at end of file
```

Chapter 3

Internship Task 2: Integration of Compression Force Support in Bolted Connection Modules

This module was developed by Nishi Kant Mandal (`https://github.com/nishikantmandal007/Osdag/tree/feat/Compression_force`) during the fellowship period.

3.1 Task 2: Problem Statement

The objective of this task was to extend the existing **Bolted Butt Joint** and **Bolted Lap Joint** design modules to support both tension and compression loading conditions. Similar to the welded connections, these bolted connection modules were originally designed only for tensile forces. Bolted connections in compression members (e.g., column splices, strut connections) behave differently than in tension. Key differences include the treatment of the net section area—bolt holes reduce tensile capacity due to rupture but do not reduce compressive capacity—and the absence of block shear failure in compression. The goal was to implement a comprehensive design approach that handles both load types, strictly adhering to IS 800:2007 (Clause 6 for tension, Clause 7 for compression), while providing transparency into the failure modes of both the bolts and the connected plates.

3.2 Task 2: Tasks Done

The integration required comprehensive updates to the user interface and the core verification logic for both bolted connection modules. A new base metal strength verification function was introduced to handle the complexities of tension vs. compression checks.

3.2.1 Subtask 2.1: Bolted Butt Joint Modifications

The **Butt Joint Bolted** module (`butt_joint_bolted.py`) was enhanced to include base metal checks and compression support.

User Interface Updates (Butt Joint)

- **Design Preference:** Added "Design" tab with "Design For" (Tension/Compression).
- **Force Input:** `KEY_TENSILE_FORCE` replaced with `KEY_AXIAL_FORCE`. Automatic detection of compression mode for negative limit state inputs.
- **Detailed Output:** Added breakdown of utilization ratios for Bolts and Base Metal separately.

Design Logic Changes (Butt Joint)

A new method `check_base_metal_strength` was implemented. It calculates capacity based on the design mode:

- **Compression:** Capacity based on Gross Yielding (Cl. 7.1.2).
- **Tension:** Capacity based on the minimum of Gross Yielding, Net Rupture, and Block Shear.

Listing 3.1: Butt Joint Base Metal Check

```
1 def check_base_metal_strength(self):
2     # ... (omitted logging setup) ...
3     if self.design_for == 'Compression':
4         self.T_db = self.A_g * fy / self.gamma_m0
```

```
        logger.info(f": Design strength of plate in compression = {self
            .T_db / 1000:.2f} kN [Cl.7.1.2]")
    else:
        # Tension Checks
        n_holes = max(self.rows, 1)
        hole_dia = self.bolt.dia_hole if hasattr(self.bolt, 'dia_hole')
            else 0.0
        net_width = float(self.width) - n_holes * hole_dia

        # ... (Validation of net_width omitted) ...
        self.A_n = plate_thk_min * net_width
        shear_lag_factor = 0.7  # IS 800:2007 Cl.6.3.3 for butt joints

        T_dg = self.A_g * fy / self.gamma_m0
        T_dn = 0.9 * self.A_n * fu * shear_lag_factor / self.gamma_m1

        # Block Shear
        T_db_block = IS800_2007.cl_6_4_1_block_shear_strength(A_vg,
            A_vn, A_tg, A_tn, fu, fy)
        self.T_db = min(T_dg, T_dn, T_db_block)
        logger.info(f": Design strength of plate in tension = {self.
            T_db / 1000:.2f} kN [Cl.6.2.2, 6.2.3, 6.3.3]")
```

Note: A shear lag factor of 0.7 is applied for butt joints as per general practice implemented in the module logic.

3.2.2 Subtask 2.2: Bolted Lap Joint Modifications

The **Lap Joint Bolted** module (`lap_joint_bolted.py`) received parallel updates.

User Interface Updates (Lap Joint)

Updates mirror those in the butt joint module: "Design" tab, "Axial Force" input, and detailed utilization breakdown.

Design Logic Changes (Lap Joint)

The `check_base_metal_strength` logic is similar but specifically tailored for lap joints (e.g., shear lag factor application).

Listing 3.2: Lap Joint Base Metal Check

```
def check_base_metal_strength(self):
    # ... (omitted logging setup) ...
    if self.design_for == 'Compression':
        self.T_db = self.A_g * fy / self.gamma_m0
        logger.info(f": Design strength of plate in compression = {self
            .T_db / 1000:.2f} kN [Cl.7.1.2]")
    else:
        # Tension Checks with Shear Lag
        n_holes = max(self.rows, 1)
        # ... (net width calc) ...

        self.A_n = plate_thk_min * net_width
        shear_lag_factor = 0.7  # IS 800:2007 Cl.6.3.3 for lap joints

        T_dg = self.A_g * fy / self.gamma_m0
        T_dn = 0.9 * self.A_n * fu * shear_lag_factor / self.gamma_m1

        # Block Shear Check included
        # ...
        T_db_block = IS800_2007.cl_6_4_1_block_shear_strength(A_vg,
            A_vn, A_tg, A_tn, fu, fy)
        self.T_db = min(T_dg, T_dn, T_db_block)
```

3.2.3 Summary of Impact

The updates provide a robust framework for designing bolted connections under compression.

- **Utilization Breakdown:** Users can now see whether the Bolt Capacity or the Base Metal Capacity is the governing factor.
- **Automatic Detection:** The module efficiently handles negative inputs by auto-switching to compression mode.
- **Comprehensive Checks:** The addition of the base metal check (Block Shear, Net Rupture, Gross Yielding) fills a critical gap in the previous implementation.

3.3 Task 2: Documentation

Explicit documentation of the logic changes is embedded in the source code. The updated design reports clearly distinguish between "Base Metal Capacity" (referencing Cl. 7.1.2 for compression) and "Bolt Capacity," providing users with a transparent verification of their design against IS 800:2007 standards.

3.3.1 Full code

The following command imports the entire source code for the module directly into the report for complete reference. This assumes the file `butt_joint_bolted_diff.py` is placed in a `codes` sub-directory.

```
diff --git a/src/osdag/design_type/connection/butt_joint_bolted.
   py b/src/osdag/design_type/connection/butt_joint_bolted.py
index 9f36ff9a..6960d7a1 100644
--- a/src/osdag/design_type/connection/butt_joint_bolted.py
+++ b/src/osdag/design_type/connection/butt_joint_bolted.py
@@ -30,6 +30,12 @@ class ButtJointBolted(MomentConnection):
     def __init__(self):
         super(ButtJointBolted, self).__init__()
         self.design_status = False
+        self.design_for = 'Tension'
+        self.axial_force_kN = 0.0
+        self.axial_force = 0.0
+        self.base_metal_capacity_kN = None
+        self.utilization_breakdown = {}
+        self.design_error = ''
         self.spacing = None
         self.packing_plate_thickness = 0.0
         self.beta_pkg = 1.0
@@ -42,9 +48,10 @@ class ButtJointBolted(MomentConnection):
     ##############################################
     def tab_list(self):
         tabs = []
```

```
-         # Only Bolt and Detailing tabs
+         # Bolt, Detailing, and Design tabs
          tabs.append(("Bolt", TYPE_TAB_2, self.bolt_values))
          tabs.append(("Detailing", TYPE_TAB_2, self.
             detailing_values))
+         tabs.append(("Design", TYPE_TAB_2, self.design_values))
          return tabs

      def tab_value_changed(self):
@@ -160,6 +167,20 @@ class ButtJointBolted(MomentConnection):
      ####################################
      # Design Preference Functions End
      ####################################
+     def design_values(self, input_dictionary):
+         values = {
+             KEY_DESIGN_FOR: 'Tension'
+         }
+
+         if input_dictionary and KEY_DESIGN_FOR in
    input_dictionary:
+             values[KEY_DESIGN_FOR] = input_dictionary[
    KEY_DESIGN_FOR]
+
+         design = []
+         t1 = (KEY_DESIGN_FOR, KEY_DISP_DESIGN_FOR, TYPE_COMBOBOX
    ,
+               ['Tension', 'Compression'], values[KEY_DESIGN_FOR
    ])
+         design.append(t1)
+
+         return design

      def set_osdaglogger(key):
```

```
@@ -229,7 +250,7 @@ class ButtJointBolted(MomentConnection):
         t6 = (None, DISP_TITLE_FSL, TYPE_TITLE, None, True, 'No
            Validator')
         options_list.append(t6)

-        t17 = (KEY_TENSILE_FORCE, KEY_DISP_TENSILE_FORCE,
   TYPE_TEXTBOX, None, True, 'Int Validator')
+        t17 = (KEY_AXIAL_FORCE, KEY_DISP_AXIAL_FORCE,
   TYPE_TEXTBOX, None, True, 'Float Validator')
         options_list.append(t17)

         t9 = (None, DISP_TITLE_BOLT, TYPE_TITLE, None, True, 'No
            Validator')
@@ -335,6 +356,21 @@ class ButtJointBolted(MomentConnection):
                 self.utilization_ratio if flag else '', True)
         out_list.append(t29)

+        t30 = (KEY_OUT_DESIGN_FOR, KEY_OUT_DISP_DESIGN_FOR,
   TYPE_TEXTBOX, self.design_for if flag else '', True)
+        out_list.append(t30)
+
+        t31 = (KEY_OUT_BASE_METAL_CAPACITY,
   KEY_OUT_DISP_BASE_METAL_CAPACITY, TYPE_TEXTBOX,
+                round(self.base_metal_capacity_kN, 2) if flag
   and self.base_metal_capacity_kN is not None else '', True)
+        out_list.append(t31)
+
+        t32 = (KEY_OUT_BASE_METAL_UTILIZATION,
   KEY_OUT_DISP_BASE_METAL_UTILIZATION, TYPE_TEXTBOX,
+                self.utilization_breakdown.get('base_metal') if
   flag and self.utilization_breakdown else '', True)
+        out_list.append(t32)
+
+        t33 = (KEY_OUT_BOLT_UTILIZATION,
```

```
   KEY_OUT_DISP_BOLT_UTILIZATION, TYPE_TEXTBOX,
+                self.utilization_breakdown.get('bolt') if flag
   and self.utilization_breakdown else '', True)
+        out_list.append(t33)
+
         t20 = (KEY_OUT_BOLT_CONN_LEN, KEY_OUT_DISP_BOLT_CONN_LEN
            , TYPE_TEXTBOX,
                self.len_conn if flag else '', True)
         out_list.append(t20)
@@ -482,10 +518,31 @@ class ButtJointBolted(MomentConnection):

     def set_input_values(self, design_dictionary):
         """Initialize components required for butt joint design
            as per IS 800:2007"""
+
+        design_dictionary_with_defaults = design_dictionary.copy
   ()
+        for key in (KEY_SHEAR, KEY_AXIAL, KEY_MOMENT):
+            if key not in design_dictionary_with_defaults:
+                design_dictionary_with_defaults[key] = 0.0
+
+        super(ButtJointBolted, self).set_input_values(self,
   design_dictionary_with_defaults)
+
         self.module = design_dictionary[KEY_MODULE]
         self.mainmodule = "Butt Joint Bolted Connection"
         self.main_material = design_dictionary[KEY_MATERIAL]
-        self.tensile_force = design_dictionary[KEY_TENSILE_FORCE
   ]
+
+        self.design_for = design_dictionary.get(KEY_DESIGN_FOR,
   'Tension')
+        axial_input = design_dictionary.get(
+            KEY_AXIAL_FORCE,
```

```
+            design_dictionary.get(KEY_AXIAL,
+                                  design_dictionary.get(
    KEY_TENSILE_FORCE, 0)))
+        axial_value = float(axial_input)
+        if axial_value < 0 and KEY_DESIGN_FOR not in
    design_dictionary:
+            self.design_for = 'Compression'
+        self.axial_force_kN = abs(axial_value)
+        self.axial_force = self.axial_force_kN * 1000.0
+        # Legacy naming issue
+        self.tensile_force = self.axial_force_kN  # legacy
    naming in downstream methods
+
         self.width = design_dictionary[KEY_PLATE_WIDTH]

         # Initialize plates with material properties
@@ -859,14 +916,44 @@ class ButtJointBolted(MomentConnection):
                 self.bolt.bolt_capacity = round(self.bolt.
                    bolt_capacity, 2)

             # Calculate utilization ratio
-            bltcap = self.bolt.bolt_capacity
-            if bltcap < 1:
-                bltcap = 1
-            self.utilization_ratio = float(self.tensile_force) /
     (bltcap * self.number_bolts)
-            self.utilization_ratio = round(self.
    utilization_ratio, 2)
+            bolt_capacity_kN = self.bolt.bolt_capacity
+            bolt_capacity_total = bolt_capacity_kN * self.
    number_bolts if bolt_capacity_kN else 0.0
+            if bolt_capacity_total <= 0:
+                logger.error(": Bolt capacity is zero. Increase
    bolt size/grade or adjust layout.")
```

```
+                self.design_status = False
+                self.design_error = "Bolt capacity is zero."
+                return
+            bolt_util = self.axial_force_kN /
    bolt_capacity_total
+
+            if not self.check_base_metal_strength():
+                return
+
+            if not self.base_metal_capacity_kN or self.
    base_metal_capacity_kN <= 0:
+                logger.error(": Base metal capacity is zero or
    undefined. Check plate selection.")
+                self.design_status = False
+                self.design_error = "Base metal capacity is zero
     or undefined."
+                return
+
+            base_util = self.axial_force_kN / self.
    base_metal_capacity_kN
+
+            def _format_util(value, decimals=3):
+                if math.isinf(value) or math.isnan(value):
+                    return 'Inf'
+                return round(value, decimals)
+
+            overall_util = max(bolt_util, base_util)
+            self.utilization_breakdown = {
+                'bolt': _format_util(bolt_util),
+                'base_metal': _format_util(base_util)
+            }
+
+            if math.isinf(overall_util) or math.isnan(
    overall_util):
```

```
+                 self.utilization_ratio = 'Inf'
+             else:
+                 self.utilization_ratio = round(overall_util, 2)

             # Check if utilization ratio is less than 1 for
                valid design
-             if self.utilization_ratio >= 1:
+             if overall_util >= 1:
                 self.design_status = False
                 logger.error(": Utilization ratio is greater
                    than or equal to 1. Design is not safe.")
                 logger.info(" :==========End Of design==========
                    ")
@@ -881,6 +968,67 @@ class ButtJointBolted(MomentConnection):
             print("Max min gauge pitch dist",self.
                max_gauge_round,self.bolt.min_gauge_round, self.
                max_pitch_round, self.bolt.min_pitch_round)


+    def check_base_metal_strength(self):
+        global logger
+        try:
+            logger
+        except NameError:
+            logger = logging.getLogger('Osdag')
+
+        logger.info(": ============== Base Metal Strength Check
   ==============")
+
+        plate_thk_min = min(float(self.plate1thk), float(self.
   plate2thk))
+        fy = min(self.plate1.fy, self.plate2.fy)
+        fu = min(self.plate1.fu, self.plate2.fu)
+
```

```
+         self.gamma_m0 = 1.10
+         self.gamma_m1 = 1.25
+
+         self.A_g = plate_thk_min * float(self.width)
+
+         if self.design_for == 'Compression':
+             self.T_db = self.A_g * fy / self.gamma_m0
+             logger.info(f": Design strength of plate in
    compression = {self.T_db / 1000:.2f} kN [Cl.7.1.2]")
+         else:
+             n_holes = max(self.rows, 1)
+             hole_dia = self.bolt.dia_hole if hasattr(self.bolt,
    'dia_hole') else 0.0
+             net_width = float(self.width) - n_holes * hole_dia
+
+             if net_width <= 0:
+                 logger.error(": Net width becomes zero/negative
    after deducting bolt holes. Increase plate width or reduce
    rows.")
+                 self.design_status = False
+                 self.design_error = "Net width insufficient for
    bolt holes."
+                 return False
+
+             self.A_n = plate_thk_min * net_width
+             shear_lag_factor = 0.7  # IS 800:2007 Cl.6.3.3 for
    butt joints
+
+             T_dg = self.A_g * fy / self.gamma_m0
+             T_dn = 0.9 * self.A_n * fu * shear_lag_factor / self
    .gamma_m1
+             self.T_dg = T_dg
+             self.T_dn = T_dn
+             self.T_db = min(T_dg, T_dn)
```

```
+
+            # Calculate block shear strength
+            A_vg = plate_thk_min * ((self.rows - 1) * self.
   final_gauge + self.final_edge_dist)
+            A_vn = plate_thk_min * ((self.rows - 1) * self.
   final_gauge + self.final_edge_dist - (self.rows - 0.5) *
   hole_dia)
+            A_tg = plate_thk_min * self.final_end_dist
+            A_tn = plate_thk_min * (self.final_end_dist - 0.5 *
   hole_dia)
+
+            T_db_block = IS800_2007.
   cl_6_4_1_block_shear_strength(A_vg, A_vn, A_tg, A_tn, fu, fy)
+            self.T_db = min(self.T_db, T_db_block)
+            logger.info(f": Design strength of plate in tension
   = {self.T_db / 1000:.2f} kN [Cl.6.2.2, 6.2.3, 6.3.3]")
+
+        if self.T_db <= 0:
+            logger.error(": Plate design strength is non-
   positive. Check input dimensions/material.")
+            self.design_status = False
+            self.design_error = "Plate design strength is non-
   positive."
+            return False
+
+        self.base_metal_capacity_kN = self.T_db / 1000.0
+        return True
+
+
    def save_design(self, popup_summary):
        """
        Generate design report for Bolted Butt Joint Connection
           as per IS 800:2007
```

The following command imports the entire source code for the module directly into

the report for complete reference. This assumes the file `lap_joint_bolted_diff.py` is placed in a `codes` sub-directory.

```
diff --git a/src/osdag/design_type/connection/lap_joint_bolted.py
    b/src/osdag/design_type/connection/lap_joint_bolted.py
index f64be500..165e1d31 100644
--- a/src/osdag/design_type/connection/lap_joint_bolted.py
+++ b/src/osdag/design_type/connection/lap_joint_bolted.py
@@ -23,10 +23,18 @@ import logging

 import math

+from PyQt5.QtCore import Qt
+
 class LapJointBolted(MomentConnection):
     def __init__(self):
         super(LapJointBolted, self).__init__()
         self.design_status = False
+        self.design_for = 'Tension'
+        self.axial_force_kN = 0.0
+        self.axial_force = 0.0
+        self.base_metal_capacity_kN = None
+        self.utilization_breakdown = {}
+        self.design_error = ''

         # self.spacing = None

@@ -39,6 +47,7 @@ class LapJointBolted(MomentConnection):
         # Only Bolt and Detailing tabs
         tabs.append(("Bolt", TYPE_TAB_2, self.bolt_values))
         tabs.append(("Detailing", TYPE_TAB_2, self.
            detailing_values))
+        tabs.append(("Design", TYPE_TAB_2, self.design_values))
         return tabs
```

```
     def tab_value_changed(self):
@@ -62,6 +71,11 @@ class LapJointBolted(MomentConnection):
         design_input.append(("Detailing", TYPE_COMBOBOX, [
             KEY_DP_DETAILING_EDGE_TYPE  # For edge preparation
                method
         ]))
+
+        # Design preferences
+        design_input.append(("Design", TYPE_COMBOBOX, [
+            KEY_DESIGN_FOR
+        ]))

         return design_input

@@ -73,7 +87,8 @@ class LapJointBolted(MomentConnection):
             KEY_DP_BOLT_TYPE,
             KEY_DP_BOLT_HOLE_TYPE,
             KEY_DP_BOLT_SLIP_FACTOR,
-            KEY_DP_DETAILING_EDGE_TYPE
+            KEY_DP_DETAILING_EDGE_TYPE,
+            KEY_DESIGN_FOR
         ], ''))

         return design_input
@@ -84,10 +99,26 @@ class LapJointBolted(MomentConnection):
             KEY_DP_BOLT_TYPE: "Non Pre-tensioned",
             KEY_DP_BOLT_HOLE_TYPE: "Standard",
             KEY_DP_BOLT_SLIP_FACTOR: "0.3",
-            KEY_DP_DETAILING_EDGE_TYPE: "Sheared or hand flame
   cut"
+            KEY_DP_DETAILING_EDGE_TYPE: "Sheared or hand flame
   cut",
+            KEY_DESIGN_FOR: 'Tension'
         }
```

```
         return defaults.get(key)

+    def design_values(self, input_dictionary):
+        values = {
+            KEY_DESIGN_FOR: 'Tension'
+        }
+
+        if input_dictionary and KEY_DESIGN_FOR in
   input_dictionary:
+            values[KEY_DESIGN_FOR] = input_dictionary[
   KEY_DESIGN_FOR]
+
+        design = []
+        t1 = (KEY_DESIGN_FOR, KEY_DISP_DESIGN_FOR, TYPE_COMBOBOX
   ,
+              ['Tension', 'Compression'], values[KEY_DESIGN_FOR
   ])
+        design.append(t1)
+
+        return design
+
     def detailing_values(self, input_dictionary):
         values = {
             KEY_DP_DETAILING_EDGE_TYPE: 'Sheared or hand flame
                cut'
@@ -172,7 +203,7 @@ class LapJointBolted(MomentConnection):
         t6 = (None, DISP_TITLE_FSL, TYPE_TITLE, None, True, 'No
            Validator')
         options_list.append(t6)

-        t17 = (KEY_TENSILE_FORCE, KEY_DISP_TENSILE_FORCE,
   TYPE_TEXTBOX, None, True, 'Int Validator')
+        t17 = (KEY_AXIAL_FORCE, KEY_DISP_AXIAL_FORCE,
   TYPE_TEXTBOX, None, True, 'Float Validator')
```

```
         options_list.append(t17)

         t9 = (None, DISP_TITLE_BOLT, TYPE_TITLE, None, True, 'No
             Validator')
@@ -270,6 +301,21 @@ class LapJointBolted(MomentConnection):

         t29 = (KEY_UTILIZATION_RATIO, KEY_DISP_UTILIZATION_RATIO
             , TYPE_TEXTBOX,self.utilization_ratio if flag else ''
             , True)
         out_list.append(t29)
+
+         t30 = (KEY_OUT_DESIGN_FOR, KEY_OUT_DISP_DESIGN_FOR,
    TYPE_TEXTBOX, self.design_for if flag else '', True)
+         out_list.append(t30)
+
+         t31 = (KEY_OUT_BASE_METAL_CAPACITY,
    KEY_OUT_DISP_BASE_METAL_CAPACITY, TYPE_TEXTBOX,
+                 round(self.base_metal_capacity_kN, 2) if flag
    and self.base_metal_capacity_kN is not None else '', True)
+         out_list.append(t31)
+
+         t32 = (KEY_OUT_BASE_METAL_UTILIZATION,
    KEY_OUT_DISP_BASE_METAL_UTILIZATION, TYPE_TEXTBOX,
+                 self.utilization_breakdown.get('base_metal') if
    flag and self.utilization_breakdown else '', True)
+         out_list.append(t32)
+
+         t33 = (KEY_OUT_BOLT_UTILIZATION,
    KEY_OUT_DISP_BOLT_UTILIZATION, TYPE_TEXTBOX,
+                 self.utilization_breakdown.get('bolt') if flag
    and self.utilization_breakdown else '', True)
+         out_list.append(t33)

         t21 = (KEY_OUT_SPACING, KEY_OUT_DISP_SPACING,
```

```
            TYPE_OUT_BUTTON, ['Spacing Details', self.spacing],
            True)
        out_list.append(t21)
@@ -311,10 +357,10 @@ class LapJointBolted(MomentConnection):
                            else:
                                flag1 = True

-                    if option[2] == TYPE_TEXTBOX and option[0]
    == KEY_TENSILE_FORCE:
-
-                        if float(design_dictionary[option[0]])
    <= 0.0:
-                            error = "Input value(s) cannot be
    equal or less than zero."
+                    if option[2] == TYPE_TEXTBOX and option[0]
    == KEY_AXIAL_FORCE:
+                        axial_val = float(design_dictionary[
    option[0]])
+                        if math.isclose(axial_val, 0.0, abs_tol
    =1e-9):
+                            error = "Input value for Axial Force
     must be non-zero."
                            all_errors.append(error)
                        else:
                            flag2 = True
@@ -334,23 +380,45 @@ class LapJointBolted(MomentConnection):

      def set_input_values(self, design_dictionary):

-        "initialisation of components required to design a
    tension member along with connection"
+        "initialisation of components required to design a lap
    joint for axial load (tension/compression)"
+
```

```
+        design_dictionary_with_defaults = design_dictionary.copy
   ()
+        for key in (KEY_SHEAR, KEY_AXIAL, KEY_MOMENT):
+            if key not in design_dictionary_with_defaults:
+                design_dictionary_with_defaults[key] = 0.0
+
+        super(LapJointBolted, self).set_input_values(self,
   design_dictionary_with_defaults)

         self.module = design_dictionary[KEY_MODULE]
         self.mainmodule = "Lap Joint Bolted Connection"
         self.main_material = design_dictionary[KEY_MATERIAL]
-        self.tensile_force = design_dictionary[KEY_TENSILE_FORCE
   ]
-        self.width = design_dictionary[KEY_PLATE_WIDTH]
-        self.plate1 = Plate(thickness=[design_dictionary[
   KEY_PLATE1_THICKNESS]],
-                            material_grade=design_dictionary[
   KEY_MATERIAL],width=design_dictionary[KEY_PLATE_WIDTH])
-        self.plate2 = Plate(thickness=[design_dictionary[
   KEY_PLATE2_THICKNESS]],
-                                material_grade=design_dictionary[
   KEY_MATERIAL],width=design_dictionary[KEY_PLATE_WIDTH])
+
+        self.design_for = design_dictionary.get(KEY_DESIGN_FOR,
   'Tension')
+        axial_input = design_dictionary.get(
+            KEY_AXIAL_FORCE,
+            design_dictionary.get(KEY_AXIAL,
+                                  design_dictionary.get(
   KEY_TENSILE_FORCE, 0)))
+        axial_value = float(axial_input)
+        if axial_value < 0 and KEY_DESIGN_FOR not in
   design_dictionary:
```

```
+            self.design_for = 'Compression'
+        self.axial_force_kN = abs(axial_value)
+        self.axial_force = self.axial_force_kN * 1000.0
+        # Legacy naming issue
+        self.tensile_force = self.axial_force_kN  # legacy
    naming in downstream methods
+
+        self.width = float(design_dictionary[KEY_PLATE_WIDTH])
+        plate1_thk = float(design_dictionary[
    KEY_PLATE1_THICKNESS])
+        plate2_thk = float(design_dictionary[
    KEY_PLATE2_THICKNESS])
+        self.plate1 = Plate(thickness=[plate1_thk],
+                            material_grade=design_dictionary[
    KEY_MATERIAL], width=self.width)
+        self.plate2 = Plate(thickness=[plate2_thk],
+                            material_grade=design_dictionary[
    KEY_MATERIAL], width=self.width)
        self.bolt = Bolt(grade=design_dictionary[KEY_GRD],
           diameter=design_dictionary[KEY_D],
-                         bolt_type=design_dictionary[KEY_TYP],
-                         bolt_hole_type=design_dictionary[
    KEY_DP_BOLT_HOLE_TYPE],
-                         edge_type=design_dictionary[
    KEY_DP_DETAILING_EDGE_TYPE],
-                         mu_f=design_dictionary.get(
    KEY_DP_BOLT_SLIP_FACTOR, None),
-                         )
+                          bolt_type=design_dictionary[KEY_TYP],
+                          bolt_hole_type=design_dictionary[
    KEY_DP_BOLT_HOLE_TYPE],
+                          edge_type=design_dictionary[
    KEY_DP_DETAILING_EDGE_TYPE],
+                          mu_f=design_dictionary.get(
```

```
   KEY_DP_BOLT_SLIP_FACTOR, None),
+                          )
         self.planes = 1
         self.count = 0
         self.slip_res = None
@@ -371,6 +439,9 @@ class LapJointBolted(MomentConnection):
         self.utilization_ratio = 0
         self.bij = 0
         self.blg = 0
+        self.base_metal_capacity_kN = None
+        self.utilization_breakdown = {}
+        self.design_error = ''
         self.select_bolt_dia_and_grade(self,design_dictionary)

     def select_bolt_dia_and_grade(self,design_dictionary):
@@ -576,22 +647,18 @@ class LapJointBolted(MomentConnection):


     def final_formatting(self,design_dictionary):
-        # print("I am herefa fafjafjafjafjajfjafajf")
-        # print(self.bolt)
-
-        gauge_dist = (float(self.width) - 2*self.bolt.
   min_end_dist_round)/(self.rows - 1)
+        gauge_divisor = max(self.rows - 1, 1)
+        gauge_dist = (float(self.width) - 2 * self.bolt.
   min_end_dist_round) / gauge_divisor

         if gauge_dist > self.max_gauge_round:
             self.final_gauge = self.max_gauge_round
             self.final_pitch = self.bolt.min_pitch_round

-            enddist = (float(self.width) - ((self.rows - 1)*self
   .final_gauge))/2
```

```
+            enddist = (float(self.width) - ((self.rows - 1) *
   self.final_gauge)) / 2
             if enddist > self.bolt.max_end_dist_round:
                 self.design_status = False
-                self.number_r_c_bolts(self,design_dictionary
   ,0,1)
-                # print("okay")
-
-                # self.design_status = True
+                self.number_r_c_bolts(self, design_dictionary,
   0, 1)
+                return
             else:
                 self.final_end_dist = enddist
                 self.final_edge_dist = enddist
@@ -599,54 +666,130 @@ class LapJointBolted(MomentConnection):
         else:
             self.final_gauge = gauge_dist
             self.final_pitch = self.bolt.min_pitch_round
-            enddist = (float(self.width) - ((self.rows - 1)*self
   .final_gauge))/2
+            enddist = (float(self.width) - ((self.rows - 1) *
   self.final_gauge)) / 2
             if enddist > self.bolt.max_end_dist_round:
-                # self.loop_helper_func(self,design_dictionary)
-                # print("okay")
-                # self.design_status = False
                 self.design_status = False
-                self.number_r_c_bolts(self,design_dictionary
   ,0,1)
-
-
+                self.number_r_c_bolts(self, design_dictionary,
   0, 1)
```

```
+                return
             else:
                 self.final_end_dist = enddist
                 self.final_edge_dist = enddist
                 self.design_status = True
-        # print("I got here")
+
         if self.bolt.bolt_type == 'Bearing Bolt':
-            self.bolt.bolt_shear_capacity = self.bolt.
    bolt_shear_capacity/ 1000
-            self.bolt.bolt_bearing_capacity = self.bolt.
    bolt_bearing_capacity / 1000
-            self.bolt.bolt_bearing_capacity = round(self.bolt.
    bolt_bearing_capacity, 2)
-            self.bolt.bolt_shear_capacity = round(self.bolt.
    bolt_shear_capacity, 2)
-            self.bolt.bolt_capacity = self.bolt.bolt_capacity /
    1000
-            self.bolt.bolt_capacity = round(self.bolt.
    bolt_capacity, 2)
+            self.bolt.bolt_shear_capacity = round(self.bolt.
    bolt_shear_capacity / 1000, 2)
+            self.bolt.bolt_bearing_capacity = round(self.bolt.
    bolt_bearing_capacity / 1000, 2)
+            self.bolt.bolt_capacity = round(self.bolt.
    bolt_capacity / 1000, 2)
         else:
-            self.slip_res = self.slip_res / 1000
-            self.slip_res = round(self.slip_res, 2)
-            self.bolt.bolt_capacity = self.bolt.bolt_capacity /
    1000
-            self.bolt.bolt_capacity = round(self.bolt.
    bolt_capacity, 2)
-
```

```
-         # print("Going for util ratio")
-         # print("Still here")
-         # print("Numbolts",self.number_bolts)
-         bltcap = self.bolt.bolt_capacity
-         if bltcap < 1:
-             bltcap = 1
-         self.utilization_ratio = float(self.tensile_force) / (
    bltcap * self.number_bolts)
-         self.utilization_ratio = round(self.utilization_ratio,
    2)
-
-         self.final_gauge = round(self.final_gauge,0)
-         self.final_pitch = round(self.final_pitch,0)
-         # print("fafafafafa",self.final_edge_dist, self.
    final_end_dist, self.final_pitch, self.final_gauge)
-         print("FINAL FINAL",self.bolt)
-         print("Final Edge/End/Gauge/Pitch",self.final_edge_dist,
    self.final_end_dist,self.final_gauge,self.final_pitch)
-
-         # print(self)
-         # print("faahfnafanfaf")
-         print("Max and min end edge dist ",self.bolt.
    max_end_dist_round, self.bolt.min_end_dist_round, self.bolt.
    max_edge_dist_round, self.bolt.min_edge_dist_round)
-         print("Max min gauge pitch dist",self.max_gauge_round,
    self.bolt.min_gauge_round, self.max_pitch_round, self.bolt.
    min_pitch_round)
-         # self.design_status = True
+             self.slip_res = round(self.slip_res / 1000, 2)
+             self.bolt.bolt_capacity = round(self.bolt.
    bolt_capacity / 1000, 2)
+
+         bolt_capacity_kN = self.bolt.bolt_capacity
+         bolt_capacity_total = bolt_capacity_kN * self.
```

```
   number_bolts if bolt_capacity_kN else 0.0
+        if bolt_capacity_total <= 0:
+            logger.error(": Bolt capacity is zero. Increase bolt
    size/grade or adjust layout.")
+            self.design_status = False
+            self.design_error = "Bolt capacity is zero."
+            return
+        bolt_util = self.axial_force_kN / bolt_capacity_total
+
+        if not self.check_base_metal_strength():
+            return
+
+        if not self.base_metal_capacity_kN or self.
   base_metal_capacity_kN <= 0:
+            logger.error(": Base metal capacity is zero or
   undefined. Check plate selection.")
+            self.design_status = False
+            self.design_error = "Base metal capacity is zero or
   undefined."
+            return
+
+        base_util = self.axial_force_kN / self.
   base_metal_capacity_kN
+
+        def _format_util(value, decimals=3):
+            if math.isinf(value) or math.isnan(value):
+                return 'Inf'
+            return round(value, decimals)
+
+        overall_util = max(bolt_util, base_util)
+        self.utilization_breakdown = {
+            'bolt': _format_util(bolt_util),
+            'base_metal': _format_util(base_util)
+        }
```

```
+
+        if math.isinf(overall_util) or math.isnan(overall_util):
+            self.utilization_ratio = 'Inf'
+        else:
+            self.utilization_ratio = round(overall_util, 2)
+
+        self.final_gauge = round(self.final_gauge, 0)
+        self.final_pitch = round(self.final_pitch, 0)
+        self.final_end_dist = round(self.final_end_dist, 0)
+        self.final_edge_dist = round(self.final_edge_dist, 0)
+
+        print("FINAL FINAL", self.bolt)
+        print("Final Edge/End/Gauge/Pitch", self.final_edge_dist
   , self.final_end_dist, self.final_gauge, self.final_pitch)
+        print("Max and min end edge dist ", self.bolt.
  max_end_dist_round, self.bolt.min_end_dist_round, self.bolt.
  max_edge_dist_round, self.bolt.min_edge_dist_round)
+        print("Max min gauge pitch dist", self.max_gauge_round,
  self.bolt.min_gauge_round, self.max_pitch_round, self.bolt.
  min_pitch_round)
+
+
+    def check_base_metal_strength(self):
+        global logger
+        try:
+            logger
+        except NameError:
+            logger = logging.getLogger('Osdag')
+
+        logger.info(": ============== Base Metal Strength Check
  ==============")
+
+        plate_thk_min = min(float(self.plate1thk), float(self.
  plate2thk))
```

```
+        fy = min(self.plate1.fy, self.plate2.fy)
+        fu = min(self.plate1.fu, self.plate2.fu)
+
+        self.gamma_m0 = 1.10
+        self.gamma_m1 = 1.25
+
+        self.A_g = plate_thk_min * float(self.width)
+
+        if self.design_for == 'Compression':
+            self.T_db = self.A_g * fy / self.gamma_m0
+            logger.info(f": Design strength of plate in
   compression = {self.T_db / 1000:.2f} kN [Cl.7.1.2]")
+        else:
+            n_holes = max(self.rows, 1)
+            hole_dia = self.bolt.dia_hole if hasattr(self.bolt,
   'dia_hole') else 0.0
+            net_width = float(self.width) - n_holes * hole_dia
+
+            if net_width <= 0:
+                logger.error(": Net width becomes zero/negative
   after deducting bolt holes. Increase plate width or reduce
   rows.")
+                self.design_status = False
+                self.design_error = "Net width insufficient for
   bolt holes."
+                return False
+
+            self.A_n = plate_thk_min * net_width
+            shear_lag_factor = 0.7  # IS 800:2007 Cl.6.3.3 for
   lap joints
+
+            T_dg = self.A_g * fy / self.gamma_m0
+            T_dn = 0.9 * self.A_n * fu * shear_lag_factor / self
  .gamma_m1
```

```
+            self.T_dg = T_dg
+            self.T_dn = T_dn
+            self.T_db = min(T_dg, T_dn)
+
+            # Calculate block shear strength
+            A_vg = plate_thk_min * ((self.rows - 1) * self.
   final_gauge + self.final_edge_dist)
+            A_vn = plate_thk_min * ((self.rows - 1) * self.
   final_gauge + self.final_edge_dist - (self.rows - 0.5) *
   hole_dia)
+            A_tg = plate_thk_min * self.final_end_dist
+            A_tn = plate_thk_min * (self.final_end_dist - 0.5 *
   hole_dia)
+
+            T_db_block = IS800_2007.
   cl_6_4_1_block_shear_strength(A_vg, A_vn, A_tg, A_tn, fu, fy)
+            self.T_db = min(self.T_db, T_db_block)
+            logger.info(f": Design strength of plate in tension
   = {self.T_db / 1000:.2f} kN [Cl.6.2.2, 6.2.3, 6.3.3]")
+
+        if self.T_db <= 0:
+            logger.error(": Plate design strength is non-
   positive. Check input dimensions/material.")
+            self.design_status = False
+            self.design_error = "Plate design strength is non-
   positive."
+            return False

+        self.base_metal_capacity_kN = self.T_db / 1000.0
+        return True


    def call_3DColumn(self, ui, bgcolor):
```

Chapter 4

Conclusions

The fellowship provided a comprehensive and immersive experience in the practical application of structural engineering principles and software development methodologies. A significant portion of the work was dedicated to the complex task of extending Osdag's connection modules to support compression forces. This required not only a deep understanding of IS 800:2007 checks for gross section yielding and buckling but also the software engineering proficiency to integrate these checks into an existing, monolithic codebase without breaking legacy functionality.

The following sections summarize the key tasks accomplished and the skills developed during this period.

4.1 Tasks Accomplished

The focus of the internship was on extending the capabilities of Osdag's connection design modules to handle compressive loads, a critical feature for practical engineering applications.

4.2 Skills Developed

The fellowship was instrumental in cultivating a wide range of technical and professional skills, bridging the gap between academic knowledge and real-world industry application.

Table 4.1: Summary of Tasks Accomplished

Task No.	**Task Title**	**Key Accomplishments**
Task 1	Compression Force Integration (Welded)	Extended the existing Welded Butt and Lap Joint design modules to support compression loading. Implemented specific design checks for gross section yielding (Cl. 7.1.2), updated the GUI to include a design mode selector, and ensured seamless handling of both tension and compression forces.
Task 2	Compression Force Integration (Bolted)	Enhanced the Bolted Butt and Lap Joint modules to handle compressive loads. Implemented comprehensive base metal checks (including block shear and net section rupture), introduced detailed utilization breakdowns for bolts vs. base metal, and added automatic compression mode detection.

Table 4.2: Summary of Skills Developed

Skill Category	**Specific Skill**	**Application & Context**
Technical Skills	Structural Engineering Analysis	Applied IS 800:2007 standards (Cl. 6 and Cl. 7) to practical connection design, specifically differentiating between failure modes in tension and compression.
	Advanced Python Programming	Employed Object-Oriented Programming (OOP) to modify and extend existing classes within the Osdag framework, ensuring modularity and reusability.

Table 4.2: Summary of Skills Developed (Continued)

Skill Category	**Specific Skill**	**Application & Context**
	Code Auditing & Debugging	Developed a systematic approach to reviewing complex legacy codebases, identifying logical constraints, and implementing new features without regression.
	Technical Documentation	Mastered LaTeX for creating professional, high-quality technical reports documenting the integration of new design logic.
Professional Skills	Problem Solving	Demonstrated the ability to deconstruct complex engineering problems (seamlessly handling bi-directional forces) and translate them into structured software solutions.
	Attention to Detail	Practiced meticulous cross-referencing of software implementation against engineering codes (IS 800:2007) to ensure accuracy and compliance.
Re-engineering	Legacy Code Adaptation	Successfully extended complex, existing legacy modules to support new critical features (Compression Support) while maintaining backward compatibility.

Table 4.2: Summary of Skills Developed (Continued)

Skill Category	**Specific Skill**	**Application & Context**
	Logic Refactoring	Restructured monolithic design algorithms to branching logic flows, enabling a single module to dynamically adapt its verification path based on load type (Tension vs. Compression).

4.3 Summary of Experience

The successful integration of compression force support into Osdag's welded and bolted connection modules stands as a testament to the skills acquired during this fellowship. This task bridged the gap between theoretical structural analysis and practical software implementation. By refactoring the logic flow to check for both tension (rupture, block shear) and compression (yielding), the modules have been transformed into versatile design tools.

The experience highlighted the importance of modular code structure, robust error handling, and comprehensive documentation. Contributing to an open-source project like Osdag has not only refined my technical abilities in Python and LaTeX but also instilled a disciplined approach to engineering software development that prioritizes accuracy, maintainability, and user experience.

Chapter A

Appendix

This module was developed by Nishi Kant Mandal (`https://github.com/nishikantmandal007/Osdag/tree/feat/Compression_force`) during the fellowship period.

A.1 Work Reports

The following log details the part-time work undertaken during the Autumn semester (August 1, 2025 – November 30, 2025). The work was primarily focused on the research, development, and integration of compression force support for Osdag's connection modules.

Table A.1: Internship Work Report

Name	NISHI KANT MANDAL	
Project	Osdag	
Internship	Autumn Semester 2025	
Week	**Period**	**Work Description**
August 2025: Research, Analysis & Planning		

Continued on next page

Table A.1: Internship Work Report (Continued)

Week	**Period**	**Work Description**
Week 1	Aug 01 - Aug 02	**Project Initiation:** Setup of Osdag development environment. Initial review of existing Welded and Bolted connection codebases.
Week 2	Aug 04 - Aug 09	**Literature Review:** Detailed study of IS 800:2007 Clause 7 (Design for Compression Members) vs. Clause 6 (Design for Tension Members). Identified key differences in failure modes (Gross Yielding vs. Net Rupture).
Week 3	Aug 11 - Aug 16	**Architectural Planning:** Drafted the logic flow for integrating dual design modes (Tension/Compression) into single monolithic modules. Designed the new UI mockups for input parameters.
Week 4	Aug 18 - Aug 23	**Module Analysis:** In-depth analysis of `butt_joint_welded.py` and `lap_joint_welded.py`. Mapped out specific functions requiring modification for compression checks.
Week 5	Aug 25 - Aug 30	**UI Implementation Plan:** Finalized the Design Preferences dialog structure. Prepared the input validation strategy for handling negative axial forces (Compression).
September 2025: Task 1 - Welded Connections Integration		

Continued on next page

Table A.1: Internship Work Report (Continued)

Week	**Period**	**Work Description**
Week 6	Sep 01 - Sep 06	**UI Development (Welded):** Implemented the "Design" tab in the Preferences dialog. Added the "Design For" combo box and updated "Axial Force" input fields in Welded modules.
Week 7	Sep 08 - Sep 13	**Butt Joint Logic:** Implemented `check_base_metal_strength` for Butt Joint Welded. Programmed the specific check for Gross Section Yielding (P_d) for compression mode.
Week 8	Sep 15 - Sep 20	**Lap Joint Logic:** Implemented compression checks for Lap Joint Welded. Integrated logic to bypass Net Section Rupture checks when in Compression mode (Clause 7.1.2).
Week 9	Sep 22 - Sep 27	**Refactoring:** Refactored common utility functions to handle `axial_force` (absolute value) instead of `tensile_force`. Updated logger messages to reflect correct IS codes (Cl 6 vs Cl 7).
Week 10	Sep 29 - Sep 30	**Initial Testing (Welded):** Conducted unit tests on Welded Butt and Lap joints to verify that compression inputs trigger the correct design path.
October 2025: Task 2 - Bolted Connections Integration		

Continued on next page

Table A.1: Internship Work Report (Continued)

Week	**Period**	**Work Description**
Week 11	Oct 01 - Oct 04	**Bolted Module Analysis:** Analyzed `butt_joint_bolted.py` for complexity regarding net section area and bolt hole deductions in compression scenarios.
Week 12	Oct 06 - Oct 11	**UI Development (Bolted):** Ported the new UI elements (Design tab, Axial Input) to the Bolted connection modules. Ensured consistency with Welded module interfaces.
Week 13	Oct 13 - Oct 18	**Base Metal Checks:** Implemented the comprehensive `check_base_metal_strength` function for Bolted joints. Added logic to differentiate Net Area (A_n) usage for Tension vs Gross Area (A_g) usage for Compression.
Week 14	Oct 20 - Oct 25	**Advanced Checks:** Integrated Block Shear verification (Clause 6.4) specifically for Tension cases. Ensured these checks are skipped for Compression cases as per standard practice.
Week 15	Oct 27 - Oct 31	**Utilization Breakdown:** Enhanced the output logic to track and display separate utilization ratios for 'Bolt Capacity' and 'Base Metal Capacity', improving design transparency.
November 2025: Verification, Bug Fixing & Documentation		

Continued on next page

Table A.1: Internship Work Report (Continued)

Week	**Period**	**Work Description**
Week 16	Nov 01, Nov 03 - Nov 08	**Integration Testing:** Ran comprehensive test suites across all 4 modified modules. Verified automatic mode switching when entering negative capacity values.
Week 17	Nov 10 - Nov 15	**Bug Fixing:** Addressed edge cases where valid compression designs were being flagged as unsafe due to legacy shear capacity checks. Fixed minor UI glitches in the Results output window.
Week 18	Nov 17 - Nov 22	**Code Cleanup:** Removed legacy `tensile_force` references where possible. Added detailed docstrings and comments citing specific IS 800:2007 clauses for all new logic.
Week 19	Nov 24 - Nov 29	**Documentation:** Authored formal technical reports (Task 1 and Task 2) detailing the integration process, algorithmic flows, and verification results. Finalized the Internship Report.

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.