FOSSEE Winter Internship Report

On

Report Generation of Steel Connection Modules for Osdag

Submitted by

Roushan Raj

3rd Year B.Tech Student, Department of Computer Science and 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:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

February 3, 2026

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 report generation of 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.

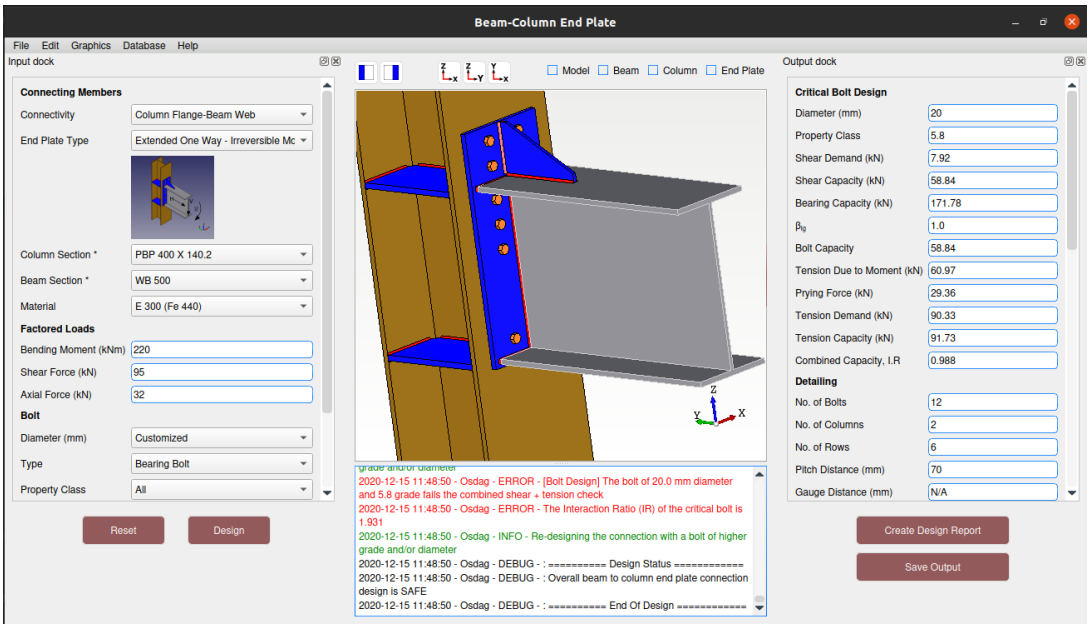

Figure 1.2: Osdag GUI

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

Report Generation of Welded Plate Girder

2.1 Problem Statement

The objective of this task was to develop an automated reporting workflow for a **Welded Plate Girder** in Osdag. The module must verify all limit states (shear capacity, moment capacity, web buckling/crippling, deflection, and weld strength) and generate a clear, stepwise LaTeX report for documentation and checking purposes.

2.2 Tasks Done

2.2.1 Design Automation Workflow

- Implemented the `save_design` function to orchestrate the report generation process, gathering all necessary design data from the PlateGirderWelded object.
- Developed helper functions `prepare_report_input` and `prepare_design_checks` to organize inputs and calculation steps into a DDCL-compliant structure.

2.2.2 Core Design Functions

- **Initialisation** (`prepare_report_input`): Extracts user inputs and derived section properties (Mass, Area, I_z, I_y, Z_p, Z_e) to populate the report header and "Section

Details" table with accurate, rounded values.

- **Design Checks** (`prepare_design_checks`):
  - **Section Classification:** Added detailed calculation logic to display flange and web outstands and their resulting classification (Plastic, Compact, Semi-compact, or Slender) in the report.
  - **Shear Capacity:** Implemented logic to dynamically display formulas for V_d and V_{cr} based on the chosen web philosophy ("Thick Web", "Simple Post Critical", or "Tension Field").
  - **Moment Capacity:** Added conditional logic to distinguish between Laterally Supported (Yielding governed) and Unsupported (Buckling governed) beams, explicitly writing out steps for M_{cr}, λ_{LT}, χ_{LT}, and f_{bd}.
  - **Stiffener Detailing:** Integrated checks for Stiffener Spacing (c/d ratios), Minimum Inertia (I_s), and Longitudinal Stiffener placement/inertia if required by the aspect ratio d/t_w.
  - **Weld Design:** Included checks for required vs. provided weld sizes for flange-web connections.
  - **Deflection:** Added Actual vs. Allowable deflection checks to the report output.
- **Final Check** (`final_format`): Implemented logic to aggregate utilization ratios (Shear, Moment, Deflection) and determine the overall "PASS/FAIL" status for the report summary.

2.2.3 Report Generation with `save_design`

- The `save_design` function is called irrespective of `design_status` is SAFE or UNSAFE. It gathers all important data (forces, material strengths, plate dimensions, weld size, cover plate type) into a `report_input` dictionary.
- It then constructs a detailed `report_check` list containing:
  - Weld size limits ($s_{\min}$, $s_{\max}$) and pass/fail status,
  - Weld strength calculation with substituted values,

  - Base metal capacity in tension or compression,
  - Overall capacity, utilization ratio and final design status.

- Mathematical expressions are wrapped using `NoEscape` so that LaTeX formulas (for example the expression of f_w or P_w) appear correctly in the final PDF.
- Finally, `CreateLatex.save_latex` is invoked to generate the formatted PDF report, including 2D/3D images and a summary table of all checks.

2.3 Python Code

The entire logic for the report generation of the Plate Girder module is within the `latex_report.py` file. Below are key snippets from the module with explanations of their functionality. Below are key snippets from the module with explanations of their functionality.

Description of the Script

The script defines the `save_design()` function and helper functions `prepare_report_input()` and `prepare_design_checks()`, which handle the translation of the PlateGirderWelded object's state into a LaTeX-compatible dictionary and list structure.

Python Code

The following snippets showcase the core logic implemented in the report generation task.

Listing 2.1: Plate Girder save_design() function

```
1 %--------------------begin code-------------
2 """
3 Module: latex_report.py
4 Description: DDCL-COMPLIANT Design Report Generator for Plate Girder (
     Updated)
5 Author: Roushan Raj
6 Date: 2026-01-16
7 """
8
9 import logging
```

```
10 from pylatex import Math
11 from pylatex.utils import NoEscape
12 from ....design_report.reportGenerator_latex import CreateLatex
13 from ....Common import KEY_DISP_SEC_PROFILE
14 from ....utils.common.Unsymmetrical_Section_Properties import
       Unsymmetrical_I_Section_Properties
15 import os
16 import math
17 from ..checks.shear import tension_field_unequal_I_corrected, calc_K_v
18 from ....utils.common.is800_2007 import IS800_2007
19
20 def save_design(popup_summary):
21     """Generate LaTeX design report for Plate Girder module"""
22     unique_logger_name = 'Osdag_plate_girder_flexure'
23     logger = logging.getLogger(unique_logger_name)
24     logger.info(" :==========Start of design report generation
           ===========")
25
26     try:
27         pg_obj = popup_summary.get('plate_girder_object')
28         if pg_obj is None:
29             logger.error("Plate Girder Object is None")
30             return False
31
32         if not pg_obj.design_status:
33             logger.warning("Design is not complete/failed checks.
                   Generating report with failures.")
34
35         report_input = prepare_report_input(pg_obj, logger)
36         report_check = prepare_design_checks(pg_obj, logger)
37
38         # Prepare report summary
39         report_summary = popup_summary.copy()
40         if 'ProfileSummary' not in report_summary:
41             report_summary['ProfileSummary'] = {
42                 'CompanyName': report_summary.get('CompanyName', ''),
43                 'CompanyLogo': report_summary.get('CompanyLogo', ''),
44                 'Group/TeamName': report_summary.get('Group/TeamName',
                       ''),
```

```
                'Designer': report_summary.get('Designer', '')
            }

        report_summary.setdefault('ProjectTitle', 'Plate Girder Design')
        report_summary.setdefault('Subtitle', 'Welded Plate Girder')
        report_summary.setdefault('JobNumber', '')
        report_summary.setdefault('Client', '')
        report_summary['does_design_exist'] = pg_obj.design_status
        report_summary.setdefault('logger_messages', '')

        # Image paths
        Disp_2d_image = []
        Disp_3D_image = "/ResourceFiles/images/3d.png"
        rel_path = os.path.abspath(".").replace("\\", "/")
        fname_no_ext = popup_summary.get("filename", "ButtJointWeldedReport")
        folder = popup_summary.get('folder', './reports')
        os.makedirs(folder, exist_ok=True)

        CreateLatex.save_latex(
            CreateLatex(), report_input, report_check,
            popup_summary, fname_no_ext, rel_path, Disp_2d_image, Disp_3D_image,
            module="Plate Girder"
        )
        logger.info(f"Report generated successfully: {fname_no_ext}.pdf")

        pdf_file_path = os.path.join(folder, f"{fname_no_ext}.pdf")
        if os.path.exists(pdf_file_path):
            file_size = os.path.getsize(pdf_file_path)
            print(f"SUCCESS: Report generated: {pdf_file_path} ({file_size} bytes)")
            return True
        else:
            print("ERROR: PDF not found")
            return False

```

```
    except Exception as e:
        logger.error(f"Report generation failed: {str(e)}")
        return False


def prepare_report_input(pg_obj, logger):
    """
    Prepare input section - ONLY user inputs, NO calculations
    """
    report_input = {}

    try:
        report_input['Module'] = 'PLATE GIRDER'
        report_input['Main Module'] = 'Flexural Member'
        # ==================== 1. General Inputs ====================
        report_input['General Inputs'] = 'TITLE'

        # Material
        material = getattr(pg_obj, 'material', None)
        report_input['Material Grade'] = getattr(material, 'designation
            ', 'E 250') if material else 'E 250'

        # Structure Type
        # Try to get from attributes, fallback to reasonable default if
             missing
        report_input['Type of Structure'] = getattr(pg_obj, '
            structure_type', 'Industrial Structure')

        # Torsional Restraint
        report_input['Torsional Restraint'] = getattr(pg_obj, '
            torsional_restraint', 'Fully Restrained')

        # Warping Restraint
        report_input['Warping Restraint'] = getattr(pg_obj, '
            warping_restraint', 'No Restraint')

        # Span
        report_input['Span (mm)'] = round(getattr(pg_obj, 'length', 0),
             1)
```

```
112
113         if hasattr(pg_obj, 'max_deflection'):
114             report_input['Maximum deflection observed (mm)'] = getattr
                (pg_obj, 'max_deflection', '-')
115
116         # ==================== Girder Properties ====================
117         girder_props = {}
118         girder_props[KEY_DISP_SEC_PROFILE] = 'ISection'
119
120         # Material Properties
121         if material:
122             girder_props['Material'] = getattr(material, 'designation',
                    'E 250')
123             girder_props['Ultimate Strength (MPa)'] = getattr(material,
                    'fu', 410)
124             girder_props['Yield Strength (MPa)'] = getattr(material, '
                fy', 250)
125             E = getattr(material, 'modulus_of_elasticity', 200000)
126             girder_props['Modulus of Elasticity (MPa)'] = E
127             mu = getattr(material, 'poisson_ratio', 0.3)
128             girder_props["Poisson's Ratio"] = mu
129             # Calculate G if not present
130             if hasattr(material, 'shear_modulus'):
131                 girder_props['Modulus of Rigidity (MPa)'] = getattr(
                        material, 'shear_modulus')
132             else:
133                 girder_props['Modulus of Rigidity (MPa)'] = round(E /
                        (2 * (1 + mu)), 2)
134
135             girder_props['Thermal Expansion'] = getattr(material, '
                coeff_thermal_expansion', 12e-6)
136
137         girder_props['Type'] = 'Welded'
138
139         # Geometry for Calculation
140         D = getattr(pg_obj, 'total_depth', 0)
141         bf_top = getattr(pg_obj, 'top_flange_width', 0)
142         bf_bot = getattr(pg_obj, 'bottom_flange_width', 0)
143         tw = getattr(pg_obj, 'web_thickness', 0)
```

```
144         tf_top = getattr(pg_obj, 'top_flange_thickness', 0)
145         tf_bot = getattr(pg_obj, 'bottom_flange_thickness', 0)
146 
147         # Section Properties Calculations
148         girder_props['Mass (kg/m)'] =
               Unsymmetrical_I_Section_Properties.calc_mass(D, bf_top,
               bf_bot, tw, tf_top, tf_bot)
149         girder_props['Area (cm$^2$)' ] = round(
               Unsymmetrical_I_Section_Properties.calc_area(D, bf_top,
               bf_bot, tw, tf_top, tf_bot) / 100, 2)
150 
151         girder_props['Moment of Area, Iz (cm$^4$)' ] = round(
               Unsymmetrical_I_Section_Properties.calc_MomentOfAreaZ(D,
               bf_top, bf_bot, tw, tf_top, tf_bot) / 10000, 2)
152         girder_props['Moment of Area, Iy (cm$^4$)' ] = round(
               Unsymmetrical_I_Section_Properties.calc_MomentOfAreaY(D,
               bf_top, bf_bot, tw, tf_top, tf_bot) / 10000, 2)
153 
154         girder_props['Radius of Gyration, rz (cm)'] = round(
               Unsymmetrical_I_Section_Properties.calc_RadiusOfGyrationZ(D,
                bf_top, bf_bot, tw, tf_top, tf_bot) / 10, 2)
155         girder_props['Radius of Gyration, ry (cm)'] = round(
               Unsymmetrical_I_Section_Properties.calc_RadiusOfGyrationY(D,
                bf_top, bf_bot, tw, tf_top, tf_bot) / 10, 2)
156 
157         girder_props['Elastic Modulus, Zez (cm$^3$)' ] = round(
               Unsymmetrical_I_Section_Properties.calc_ElasticModulusZz(D,
               bf_top, bf_bot, tw, tf_top, tf_bot) / 1000, 2)
158         girder_props['Elastic Modulus, Zey (cm$^3$)' ] = round(
               Unsymmetrical_I_Section_Properties.calc_ElasticModulusZy(D,
               bf_top, bf_bot, tw, tf_top, tf_bot) / 1000, 2)
159 
160         girder_props['Plastic Modulus, Zpz (cm$^3$)' ] = round(
               Unsymmetrical_I_Section_Properties.calc_PlasticModulusZ(D,
               bf_top, bf_bot, tw, tf_top, tf_bot) / 1000, 2)
161         girder_props['Plastic Modulus, Zpy (cm$^3$)' ] = round(
               Unsymmetrical_I_Section_Properties.calc_PlasticModulusY(D,
               bf_top, bf_bot, tw, tf_top, tf_bot) / 1000, 2)
162 
```

```
163         report_input['Girder Properties'] = girder_props
164
165         # ==================== 2. Dimensions Inputs
                ====================
166         report_input['Dimensions Inputs'] = 'TITLE'
167
168         report_input['Top Flange Width (mm)'] = round(getattr(pg_obj, '
                top_flange_width', 0), 1)
169         report_input['Top Flange Thickness (mm)'] = round(getattr(
                pg_obj, 'top_flange_thickness', 0), 1)
170         report_input['Bottom Flange Width (mm)'] = round(getattr(pg_obj
                , 'bottom_flange_width', 0), 1)
171         report_input['Bottom Flange Thickness (mm)'] = round(getattr(
                pg_obj, 'bottom_flange_thickness', 0), 1)
172         report_input['Depth of Section (mm)'] = round(getattr(pg_obj, '
                total_depth', 0), 1)
173         report_input['Web Thickness (mm)'] = round(getattr(pg_obj, '
                web_thickness', 0), 1)
174
175         # ==================== 3. Load Inputs ====================
176         report_input['Load Inputs'] = 'TITLE'
177
178         load = getattr(pg_obj, 'load', None)
179         if load:
180             report_input['Maximum Bending Moment (kN-m)'] = round(
                    getattr(load, 'moment', 0) * 1e-6, 2)
181             report_input['Maximum Shear Force (kN)'] = round(getattr(
                    load, 'shear_force', 0) * 1e-3, 2)
182         else:
183             report_input['Maximum Bending Moment (kN-m)'] = 0
184             report_input['Maximum Shear Force (kN)'] = 0
185
186         report_input['Bending moment diagram shape'] = getattr(pg_obj,
                'moment_diagram_type', 'Uniform Loading')
187
188         # ==================== 4. Support Condition Inputs
                ====================
189         report_input['Support Condition Inputs'] = 'TITLE'
190
```

```
191         report_input['Support Condition'] = getattr(pg_obj, '
                support_type', 'Major Laterally Supported')
192         report_input['Bearing length (mm)'] = round(getattr(pg_obj, '
                bearing_length', 0), 1)
193 
194         # ==================== 5. Web Philosophy Inputs
                ====================
195         report_input['Web Philosophy Inputs'] = 'TITLE'
196 
197         report_input['Web Type'] = getattr(pg_obj, 'web_philosophy', '
                Thick Web without ITS')
198 
199         # Additional Material Details (Standard)
200         if material:
201             report_input['Yield Strength, fy (MPa)'] = round(getattr(
                    material, 'fy', 0), 1)
202             report_input['Ultimate Strength, fu (MPa)'] = round(getattr
                    (material, 'fu', 0), 1)
203 
204     except Exception as e:
205         logger.error(f"Error preparing report input: {str(e)}")
206 
207     return report_input
208 
209 def prepare_design_checks(pg_obj, logger):
210     """
211     Prepare design checks - DDCL compliant with multi-line equations
212     All equations formatted per lap_joint_bolted.py pattern
213     """
214     report_check = []
215     table_format = '|p{4cm}|p{4.5cm}|p{6cm}|p{1.5cm}|'
216 
217     try:
218         # ==================== GET VALUES FROM PG_OBJ
                ====================
219         # Material
220         material = getattr(pg_obj, 'material', None)
221         fy = round(material.fy, 1) if material else 250
222         fu = round(material.fu, 1) if material else 410
```

```
223         E = getattr(material, 'modulus_of_elasticity', 200000) if
                material else 200000
224
225         # Loads
226         load = getattr(pg_obj, 'load', None)
227         M_applied = round(getattr(load, 'moment', 0) * 1e-6, 2) if load
                else 0
228         V_applied = round(getattr(load, 'shear_force', 0) * 1e-3, 2) if
                load else 0
229
230         # Dimensions
231         D = getattr(pg_obj, 'total_depth', 0)
232         d = getattr(pg_obj, 'eff_depth', 0)
233         tw = getattr(pg_obj, 'web_thickness', 0)
234         tf_top = getattr(pg_obj, 'top_flange_thickness', 0)
235         tf_bot = getattr(pg_obj, 'bottom_flange_thickness', 0)
236         bf_top = getattr(pg_obj, 'top_flange_width', 0)
237         bf_bot = getattr(pg_obj, 'bottom_flange_width', 0)
238         L = getattr(pg_obj, 'length', 0)
239
240         # Design parameters
241         gamma_m0 = getattr(pg_obj, 'gamma_m0', 1.1)
242         epsilon = getattr(pg_obj, 'epsilon', 1.0)
243
244         # Design decisions from pg_obj
245         section_class = getattr(pg_obj, 'section_class', None)
246         if section_class is None:
247             section_class = getattr(pg_obj, 'section_classification_val
                    ', 'NA')
248
249         design_status = getattr(pg_obj, 'design_status', False)
250         shear_flag1 = getattr(pg_obj, 'shearflag1', False)
251         shear_flag2 = getattr(pg_obj, 'shearflag2', False)
252         shear_flag3 = getattr(pg_obj, 'shearflag3', False)
253         moment_checks = getattr(pg_obj, 'momentchecks', False)
254         defl_check = getattr(pg_obj, 'defl_check', False)
255         web_philosophy = getattr(pg_obj, 'web_philosophy', '')
256
257         # ==================== SECTION CLASSIFICATION
```

```
            ====================
        report_check.append(['SubSection', 'Section Classification',
            table_format])

        # Top flange slenderness
        if bf_top > 0 and tf_top > 0:
            btf_top = round((bf_top - tw) / (2 * tf_top), 2)

            btf_eq = Math(inline=True)
            btf_eq.append(NoEscape(r'\begin{aligned}\\'))
            btf_eq.append(NoEscape(rf'\dfrac{{b_f - t_w}}{{2 t_f}} &= \
                dfrac{{{bf_top:.1f} - {tw:.1f}}}{{2 \times {tf_top:.1f
                }}}\\'))
            btf_eq.append(NoEscape(rf'&= {btf_top:.2f}\\'))
            btf_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Table
                2]}\\'))
            btf_eq.append(NoEscape(r'\end{aligned}'))

            # Determine Class
            outstand_top = (bf_top - tw) / 2
            class_top = IS800_2007.Table2_i(outstand_top, tf_top, fy, '
                Welded')[0]

            report_check.append([
                'Top Flange Slenderness Ratio',
                '',
                btf_eq,
                class_top
            ])

        # Bottom flange slenderness
        if bf_bot > 0 and tf_bot > 0:
            btf_bot = round((bf_bot - tw) / (2 * tf_bot), 2)

            btf_bot_eq = Math(inline=True)
            btf_bot_eq.append(NoEscape(r'\begin{aligned}\\'))
            btf_bot_eq.append(NoEscape(rf'\dfrac{{b_f - t_w}}{{2 t_f}}
                &= \dfrac{{{bf_bot:.1f} - {tw:.1f}}}{{2 \times {tf_bot
                :.1f}}}\\'))
```

```
            btf_bot_eq.append(NoEscape(rf'&= {btf_bot:.2f}\\'))
            btf_bot_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007,
                Table 2]}\\'))
            btf_bot_eq.append(NoEscape(r'\end{aligned}'))

            # Determine Class
            outstand_bot = (bf_bot - tw) / 2
            class_bot = IS800_2007.Table2_i(outstand_bot, tf_bot, fy, '
                Welded')[0]

            report_check.append([
                'Bottom Flange Slenderness Ratio',
                '',
                btf_bot_eq,
                class_bot
            ])

        # Web slenderness
        d_tw_ratio = round(d / tw, 2) if tw > 0 else 0

        dtw_eq = Math(inline=True)
        dtw_eq.append(NoEscape(r'\begin{aligned}\\'))
        dtw_eq.append(NoEscape(rf'\dfrac{{d}}{{t_w}} &= \dfrac{{{d:.1f
            }}}{{{tw:.1f}}}\\'))
        dtw_eq.append(NoEscape(rf'&= {d_tw_ratio:.2f}\\'))
        dtw_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Table 2]}\\'
            ))
        dtw_eq.append(NoEscape(r'\end{aligned}'))

        web_depth = d
        class_web = IS800_2007.Table2_iii(web_depth, tw, fy)

        report_check.append([
            'Web Slenderness Ratio',
            '',
            dtw_eq,
            class_web
        ])

```

```
        # Overall section classification
        overall_eq = Math(inline=True)
        overall_eq.append(NoEscape(r'\begin{aligned}\\'))
        overall_eq.append(NoEscape(r'&\text{Governing classification based on}\\'))
        overall_eq.append(NoEscape(r'&\text{most critical element}\\'))
        overall_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Table 2]}\\'))
        overall_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'Overall Section Classification',
            overall_eq,
            str(section_class),
            ''
        ])

        # ==================== SHEAR CAPACITY CHECK ====================
        report_check.append(['SubSection', 'Shear Capacity Check', table_format])

        V_d_N = getattr(pg_obj, 'V_d', 0)
        V_d = round(V_d_N / 1000, 2) if V_d_N else 0

        # Shear area
        Avw = round(d * tw, 2)

        avw_eq = Math(inline=True)
        avw_eq.append(NoEscape(r'\begin{aligned}\\'))
        avw_eq.append(NoEscape(rf'A_{{vw}} &= d \times t_w\\'))
        avw_eq.append(NoEscape(rf'&= {d:.1f} \times {tw:.1f}\\'))
        avw_eq.append(NoEscape(rf'&= {Avw:.2f} \text{{ mm}}^2\\'))
        avw_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl.8.4]}\\'))
        avw_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            NoEscape(r'Shear Area'),
```

```
            '',
            avw_eq,
            ''
        ])

        # Design shear strength
        if web_philosophy == 'Thick Web without ITS':
            vd_eq = Math(inline=True)
            vd_eq.append(NoEscape(r'\begin{aligned}\\'))
            vd_eq.append(NoEscape(r'V_d &= \dfrac{A_{vw} \times f_y}{\
                sqrt{3} \times \gamma_{m0}}\\'))
            vd_eq.append(NoEscape(rf'&= \dfrac{{{Avw:.2f} \times {fy:.1
                f}}}{{\sqrt{{3}} \times {gamma_m0}}}\\'))
            vd_eq.append(NoEscape(rf'&= {V_d:.2f} \text{{ kN}}\\'))
            vd_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                .8.4.1]}\\'))
            vd_eq.append(NoEscape(r'\end{aligned}'))

            shear_status = 'Pass' if shear_flag1 else 'Fail'

            report_check.append([
                NoEscape(r'Design Shear Strength'),
                f'{V_applied:.2f}',
                vd_eq,
                shear_status
            ])
        else:
            # Thin web - Check if Tension Field or Simple Post Critical
            shear_method = getattr(pg_obj, 'x', 'Simple Post Critical')

            # Common parameters for both methods
            d_eff = getattr(pg_obj, 'eff_depth', 0)
            c_val = getattr(pg_obj, 'c', 0)
            if c_val == 'NA': c_val = 0

            kv = calc_K_v(c_val, d_eff, web_philosophy)

            kv_nm = 5.35
            kv_m = 4.0
```

```
            a_ratio = c_val / d_eff if d_eff != 0 else 0

            kv_calc_eq = Math(inline=True)
            kv_calc_eq.append(NoEscape(r'\begin{aligned}\\'))
            kv_calc_eq.append(NoEscape(rf'\dfrac{{c}}{{d}} &= \dfrac{{{
                c_val:.1f}}}{{{d_eff:.1f}}} = {a_ratio:.2f}\\\\'))

            if a_ratio < 1.0:
                 kv_calc_eq.append(NoEscape(r'k_v &= 4.0 + \dfrac
                    {5.35}{(c/d)^2}\\\\'))
                 kv_calc_eq.append(NoEscape(rf'&= 4.0 + \dfrac
                    {{5.35}}{{({a_ratio:.2f})^2}} = {kv:.3f}\\'))
            else:
                 kv_calc_eq.append(NoEscape(r'k_v &= 5.35 + \dfrac
                    {4.0}{(c/d)^2}\\\\'))
                 kv_calc_eq.append(NoEscape(rf'&= 5.35 + \dfrac
                    {{4.0}}{{({a_ratio:.2f})^2}} = {kv:.3f}\\'))

            kv_calc_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                .8.4.2.2]}\\'))
            kv_calc_eq.append(NoEscape(r'\end{aligned}'))

            report_check.append([
                NoEscape(r'Buckling Coefficient'),
                '',
                kv_calc_eq,
                ''
            ])

            # Calculate Tau_crc, Lambda_w, Tau_b generically first
                using IS800 module logic
            # (replicating shear.py logic for consistency)
            mu = 0.3
            tau_crc = IS800_2007.cl_8_4_2_2_tau_crc_Simple_postcritical
                (kv, E, mu, d_eff, tw)
            lambda_w = IS800_2007.
                cl_8_4_2_2_lambda_w_Simple_postcritical(fy, tau_crc)
            tau_b = IS800_2007.cl_8_4_2_2_tau_b_Simple_postcritical(
                lambda_w, fy)
```

```
423
424             # Display Lambda_w
425             lambda_eq = Math(inline=True)
426             lambda_eq.append(NoEscape(r'\begin{aligned}\\'))
427             lambda_eq.append(NoEscape(r'\lambda_w &= \sqrt{\dfrac{f_{yw
                    }}{\sqrt{3} \tau_{cr,e}}}\\\\'))
428             lambda_eq.append(NoEscape(rf'&= \sqrt{{\dfrac{{{fy:.2f
                    }}}{{\sqrt{{3}} \times {tau_crc:.2f}}}}}\\\\'))
429             lambda_eq.append(NoEscape(rf'&= {lambda_w:.3f}\\'))
430             lambda_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                    .8.4.2.2]}\\'))
431             lambda_eq.append(NoEscape(r'\end{aligned}'))
432
433             report_check.append([
434                 NoEscape(r'Web Slenderness'),
435                 '',
436                 lambda_eq,
437                 ''
438             ])
439
440             if lambda_w <= 0.8:
441                 tau_b = round(fy / (3**0.5), 2)
442                 tau_b_eq = Math(inline=True)
443                 tau_b_eq.append(NoEscape(r'\begin{aligned}\\'))
444                 tau_b_eq.append(NoEscape(rf'\lambda_w \leq 0.8: \quad \
                        tau_b &= \dfrac{{f_{{yw}}}}{{\sqrt{{3}}}}\\\\'))
445                 tau_b_eq.append(NoEscape(rf'&= \dfrac{{{fy:.1f}}}{{\
                        sqrt{{3}}}}\\\\'))
446                 tau_b_eq.append(NoEscape(rf'&= {tau_b:.2f} \text{{ MPa
                        }}\\\\'))
447                 tau_b_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                        .8.4.2.2, Eq. 1.21]}\\'))
448                 tau_b_eq.append(NoEscape(r'\end{aligned}'))
449             elif 0.8 < lambda_w < 1.2:
450                 tau_b = round((1 - 0.8 * (lambda_w - 0.8)) * (fy /
                        (3**0.5)), 2)
451                 tau_b_eq = Math(inline=True)
452                 tau_b_eq.append(NoEscape(r'\begin{aligned}\\'))
453                 tau_b_eq.append(NoEscape(r'0.8 < \lambda_w < 1.2: \quad
```

```
                \tau_b &= [1 - 0.8(\lambda_w - 0.8)]\dfrac{f_{yw
                }}{\sqrt{3}}\\\\'))
454             tau_b_eq.append(NoEscape(rf'&= [1 - 0.8({lambda_w:.3f}
                - 0.8)] \times \dfrac{{{fy:.1f}}}{{\sqrt{{3}}}}\\\\'
                ))
455             tau_b_eq.append(NoEscape(rf'&= {tau_b:.2f} \text{{ MPa
                }}\\\\'))
456             tau_b_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                .8.4.2.2, Eq. 1.21]}\\'))
457             tau_b_eq.append(NoEscape(r'\end{aligned}'))
458         else:
459             tau_b = round(fy / (3**0.5 * lambda_w), 2)
460             tau_b_eq = Math(inline=True)
461             tau_b_eq.append(NoEscape(r'\begin{aligned}\\'))
462             tau_b_eq.append(NoEscape(r'\lambda_w \geq 1.2: \quad \
                tau_b &= \dfrac{f_{yw}}{\sqrt{3}\lambda_w}\\\\'))
463             tau_b_eq.append(NoEscape(rf'&= \dfrac{{{fy:.1f}}}{{\
                sqrt{{3}} \times {lambda_w:.3f}}}\\\\'))
464             tau_b_eq.append(NoEscape(rf'&= {tau_b:.2f} \text{{ MPa
                }}\\\\'))
465             tau_b_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                .8.4.2.2]}\\'))
466             tau_b_eq.append(NoEscape(r'\end{aligned}'))
467
468         report_check.append([
469             'Design Shear Stress',
470             '',
471             tau_b_eq,
472             ''
473         ])
474
475         V_cr_val = round(Avw * tau_b / 1000, 2)
476
477         Vcr_eq = Math(inline=True)
478         Vcr_eq.append(NoEscape(r'\begin{aligned}\\'))
479         Vcr_eq.append(NoEscape(r'V_{cr} &= A_{vw} \times \tau_b\\\\
            '))
480         Vcr_eq.append(NoEscape(rf'&= {Avw:.2f} \times {tau_b:.2f
            }\\\\'))
```

```
481             Vcr_eq.append(NoEscape(rf'&= {V_cr_val:.2f} \text{{ kN
                    }}\\\\'))
482             Vcr_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                    .8.4.2.1]}\\'))
483             Vcr_eq.append(NoEscape(r'\end{aligned}'))
484 
485             report_check.append([
486                 'Shear Buckling Resistance',
487                 '',
488                 Vcr_eq,
489                 ''
490             ])
491 
492             if shear_method == 'Tension Field Action':
493                 # --- TENSION FIELD ACTION REPORTING ---
494 
495                 # Recalculate Tension Field parameters
496                 V_cr = IS800_2007.cl_8_4_2_2_Vcr_Simple_postcritical(
                        tau_b, Avw)
497 
498                 # We need M_applied (calculated/user) but for
                        calculation of N_f, shear.py uses load.moment (N-mm)
499                 # Ensure units are correct. M_applied earlier is in kNm
                        .
500                 # tension_field_unequal_I_corrected expects M in N-mm
501                 M_design_Nmm = M_applied * 1e6
502 
503                 Nf_val = M_design_Nmm / (d_eff + (tf_top + tf_bot) / 2)
504 
505                 phi, Mfr_t, Mfr_b, s_t, s_b, w_tf, psi, fv, V_tf_val =
                        tension_field_unequal_I_corrected(
506                     c_val, d_eff, tw, fy, bf_top, tf_top, bf_bot,
                            tf_bot, Nf_val, gamma_m0, Avw, tau_b
507                 )
508 
509                 V_tf_final = round(V_tf_val / 1000, 2)
510 
511                 # 1. Tension Field Angle (phi) - DDCL Eq 1.29
512                 phi_eq = Math(inline=True)
```

```
513                 phi_eq.append(NoEscape(r'\begin{aligned}\\'))
514                 phi_eq.append(NoEscape(r'\phi &= \tan^{-1} \left( \
                        dfrac{d}{c} \right)\\\\'))
515                 phi_eq.append(NoEscape(rf'&= \tan^{{-1}} \left( \dfrac
                        {{{d_eff:.1f}}}{{{c_val:.1f}}} \right)\\\\'))
516                 phi_eq.append(NoEscape(rf'&= {phi:.2f}^\circ\\'))
517                 phi_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                        .8.4.2.2a]}\\'))
518                 phi_eq.append(NoEscape(r'\end{aligned}'))
519
520                 report_check.append([
521                     NoEscape(r'Tension Field Angle'),
522                     '',
523                     phi_eq,
524                     ''
525                 ])
526
527                 # 2. Width of Tension Field (w_tf) - DDCL Eq 1.30
528                 # w_tf = d*cos(phi) + (c - sc - st)*sin(phi)
529                 # Note: 'sc' in code is 's_b' (anchor length bottom/
                        compression?) or s_t/s_b generically
530                 # The function returns s_t (top) and s_b (bottom). DDCL
                         uses sc and st.
531                 # Just show the formula with values.
532
533                 wtf_eq = Math(inline=True)
534                 wtf_eq.append(NoEscape(r'\begin{aligned}\\'))
535                 wtf_eq.append(NoEscape(r'w_{tf} &= d \cos \phi + (c -
                        s_c - s_t) \sin \phi\\\\'))
536                 wtf_eq.append(NoEscape(rf'&= {d_eff:.1f} \cos({phi:.2f
                        }) + ({c_val:.1f} - {s_b:.1f} - {s_t:.1f}) \sin({phi
                        :.2f})\\\\'))
537                 wtf_eq.append(NoEscape(rf'&= {w_tf:.2f} \text{{ mm}}\\'
                        ))
538                 wtf_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                        .8.4.2.2a]}\\'))
539                 wtf_eq.append(NoEscape(r'\end{aligned}'))
540
541                 report_check.append([
```

```
                    NoEscape(r'Width of Tension Field'),
                    '',
                    wtf_eq,
                    ''
                ])

                # 3. Yield Strength of Tension Field (fv) - DDCL Eq 1.31
                # fv = sqrt(fy^2 - 3*tau_b^2 + psi^2) - psi
                # psi = 1.5 * tau_b * sin(2phi)

                fv_eq = Math(inline=True)
                fv_eq.append(NoEscape(r'\begin{aligned}\\'))
                fv_eq.append(NoEscape(r'\psi &= 1.5 \tau_b \sin(2\phi) \\\\'))
                fv_eq.append(NoEscape(rf'&= 1.5 \times {tau_b:.2f} \times \sin(2 \times {phi:.2f}) = {psi:.2f}\\\\'))
                fv_eq.append(NoEscape(r'f_v &= \sqrt{f_{yw}^2 - 3 \tau_b^2 + \psi^2} - \psi\\\\'))
                fv_eq.append(NoEscape(rf'&= \sqrt{{{fy:.1f}^2 - 3({tau_b:.2f})^2 + ({psi:.2f})^2}} - {psi:.2f}\\\\'))
                fv_eq.append(NoEscape(rf'&= {fv:.2f} \text{{ MPa}}\\'))
                fv_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl .8.4.2.2a]}\\'))
                fv_eq.append(NoEscape(r'\end{aligned}'))

                report_check.append([
                    NoEscape(r'Yield Strength of Web'),
                    '',
                    fv_eq,
                    ''
                ])

                # 4. Design Shear Strength (V_tf) - DDCL Eq 1.28
                vtf_eq = Math(inline=True)
                vtf_eq.append(NoEscape(r'\begin{aligned}\\'))
                vtf_eq.append(NoEscape(r'V_{tf} &= A_{vm} \tau_b + 0.9 w_{tf} t_w f_v \sin \phi\\\\')) # A_vm typically Avw
                vtf_eq.append(NoEscape(rf'&= {Avw:.2f} \times {tau_b:.2
```

```
                    f} + 0.9 \times {w_tf:.2f} \times {tw:.1f} \times {
                    fv:.2f} \times \sin({phi:.2f})\\\\'))
574                 vtf_eq.append(NoEscape(rf'&= {V_tf_final:.2f} \text{{
                    kN}}\\'))
575                 vtf_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                    .8.4.2.2a]}\\'))
576                 vtf_eq.append(NoEscape(r'\end{aligned}'))
577
578                 shear_status = 'Pass' if shear_flag1 else 'Fail'
579
580                 report_check.append([
581                     NoEscape(r'Design Shear Strength'),
582                     f'{V_applied:.2f}', # Applied Load
583                     vtf_eq,
584                     shear_status
585                 ])
586
587         # Web crippling
588         if hasattr(pg_obj, 'F_q') and pg_obj.F_q not in [None, 'NA',
                0]:
589             Fq = round(pg_obj.F_q / 1000, 2)
590             b1 = getattr(pg_obj, 'b1', 0)
591             bearing_note = ""
592             if b1 <= 0 or b1 < pg_obj.web_thickness * 2:
593                 bearing_note = " (min. assumed)"
594
595             fq_eq = Math(inline=True)
596             fq_eq.append(NoEscape(r'\begin{aligned}\\'))
597
598             # Calculate n2 for display purpose if not explicitly
                    available, based on Fq equation
599             # Fq = (b1 + n2) * tw * fy / gamma_m0
600             # (b1 + n2) = Fq * gamma_m0 / (tw * fy)
601             # n2 = [Fq * 1000 * gamma_m0 / (tw * fy)] - b1
602             # Note: Fq in pg_obj is in N. b1 in mm.
603
604             n2_disp = 0
605             if tw > 0 and fy > 0:
606                 try:
```

```
                    n2_disp = (pg_obj.F_q * gamma_m0) / (tw * fy) - b1
                except Exception:
                    n2_disp = 0

            fq_eq.append(NoEscape(r'F_q &= \dfrac{(b_1 + n_2) t_w f_y
                }{\gamma_{m0}}\\\\'))
            fq_eq.append(NoEscape(rf'&= \dfrac{{({b1:.1f} + {n2_disp:.1
                f}) \times {tw:.1f} \times {fy:.1f}}}{{{gamma_m0}}}\\\\'
                ))
            fq_eq.append(NoEscape(rf'&= {Fq:.2f} \text{{ kN}}\\'))
            fq_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                .8.7.1.1]}\\'))
            fq_eq.append(NoEscape(r'\end{aligned}'))

            fq_status = 'Pass' if shear_flag3 else 'Fail'

            report_check.append([
                NoEscape(r'Web Crippling Strength'),
                '',
                fq_eq,
                fq_status
            ])

        # ==================== MOMENT CAPACITY CHECK
            ====================
        report_check.append(['SubSection', 'Moment Capacity Check',
            table_format])

        # READ Md from pg_obj
        Md_val = getattr(pg_obj, 'Md', 0)
        Md = round(Md_val * 1e-6, 2) if Md_val and Md_val > 0 else 0

        Zp_val = getattr(pg_obj, 'plast_sec_mod_z', 0)
        Zp = round(Zp_val * 1e-3, 2) if Zp_val and Zp_val > 0 else 0

        Ze_val = getattr(pg_obj, 'elast_sec_mod_z', 0)
        Ze = round(Ze_val * 1e-3, 2) if Ze_val and Ze_val > 0 else 0

        beta_b_val = getattr(pg_obj, 'betab', 1.0)
```

```
        beta_b = round(beta_b_val, 3) if beta_b_val else 1.0

        if section_class in ['Plastic', 'Compact']:
            Z_used = Zp
            Z_label = 'Z_p'
        else:
            Z_used = Ze
            Z_label = 'Z_e'

        # Beta_b
        beta_eq = Math(inline=True)
        beta_eq.append(NoEscape(r'\begin{aligned}\\'))
        beta_eq.append(NoEscape(rf'\beta_b &= {beta_b:.2f}\\'))
        beta_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            NoEscape(r'Betab Factor'),
            '',
            beta_eq,
            ''
        ])

        # Design moment capacity
        md_eq = Math(inline=True)
        md_eq.append(NoEscape(r'\begin{aligned}\\'))

        support_type = getattr(pg_obj, 'support_type', '')
        if support_type == 'Major Laterally Unsupported':
            md_eq.append(NoEscape(r'M_d &= \beta_b Z_p f_{bd}\\\\'))
            md_eq.append(NoEscape(rf'&= {beta_b} \times {Z_used:.2f} \
                times f_{{bd}}\\\\'))
        else:
            md_eq.append(NoEscape(r'M_d &= \dfrac{\beta_b Z_p f_y}{\
                gamma_{m0}}\\'))
            md_eq.append(NoEscape(rf'&= \dfrac{{{beta_b} \times {Z_used
                :.2f} \times {fy}}}{{{gamma_m0}}}\\\\'))

        md_eq.append(NoEscape(rf'&= {Md:.2f} \text{{ kN-m}}\\\\'))

```

```
676         if support_type == 'Major Laterally Unsupported':
677             md_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                .8.2.2]}\\'))
678         else:
679             md_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                .8.2.1]}\\'))
680 
681         md_eq.append(NoEscape(r'\end{aligned}'))
682 
683         moment_status = 'Pass' if moment_checks else 'Fail'
684 
685         report_check.append([
686             NoEscape(r'Design Moment Capacity'),
687             NoEscape(rf'{M_applied:.2f}\text{{ kN-m}}'),
688             md_eq,
689             moment_status
690         ])
691 
692         # ==================== LATERAL TORSIONAL BUCKLING
                ====================
693         support_type = getattr(pg_obj, 'support_type', '')
694         if support_type in ['Major Laterally Unsupported', 'Minor
                Laterally Unsupported']:
695             report_check.append(['SubSection', 'Lateral Torsional
                    Buckling Check', table_format])
696 
697             L_eff = getattr(pg_obj, 'effective_length', 0)
698             Mcr_val = getattr(pg_obj, 'M_cr', 0)
699             Mcr = round(Mcr_val * 1e-6, 2) if Mcr_val and Mcr_val > 0
                    else 'NA'
700             lambda_LT = round(getattr(pg_obj, 'lambda_lt', 0), 3) if
                    hasattr(pg_obj, 'lambda_lt') else 'NA'
701             chi_LT = round(getattr(pg_obj, 'X_lt', 1.0), 3) if hasattr(
                    pg_obj, 'X_lt') else 'NA'
702             fbd = round(getattr(pg_obj, 'fbd_lt', 0), 2) if hasattr(
                    pg_obj, 'fbd_lt') else 'NA'
703 
704             # Effective length
705             leff_eq = Math(inline=True)
```

```
            leff_eq.append(NoEscape(r'\begin{aligned}\\'))
            leff_eq.append(NoEscape(rf'L_{{LT}} &= {L_eff:.1f} \text{{
                mm}}\\'))
            leff_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Table
                14]}\\'))
            leff_eq.append(NoEscape(r'\end{aligned}'))

            report_check.append([
                NoEscape(r'Effective Length'),
                '',
                leff_eq,
                ''
            ])

            # Critical moment
            if Mcr != 'NA':
                # Calculate properties for Mcr breakdown
                try:
                    Iy = Unsymmetrical_I_Section_Properties.
                        calc_MomentOfAreaY(D, bf_top, bf_bot, tw, tf_top
                        , tf_bot)
                    # Mcr,z (Euler) in kNm
                    # Mcr_z = (pi^2 * E * Iy) / L_LT^2
                    if L_eff > 0:
                        Mcr_z_Nmm = (math.pi**2 * E * Iy) / (L_eff**2)
                        Mcr_z = Mcr_z_Nmm * 1e-6
                    else:
                        Mcr_z = 0

                    # Back-calculate Mcr,T -> Mcr = sqrt(Mcr_z * Mcr_T)
                        => Mcr_T = Mcr^2 / Mcr_z
                    if Mcr_z > 0:
                         Mcr_T = (Mcr)**2 / Mcr_z
                    else:
                         Mcr_T = 0
                except Exception as m_e:
                    # Fallback if calc fails
                    Mcr_z = 0
                    Mcr_T = 0
```

```
740
741                 mcr_eq = Math(inline=True)
742                 mcr_eq.append(NoEscape(r'\begin{aligned}\\'))
743                 mcr_eq.append(NoEscape(r'M_{cr} &= \sqrt{M_{cr,z} \
                        times M_{cr,T}}\\'))
744                 if Mcr_z > 0 and Mcr_T > 0:
745                     mcr_eq.append(NoEscape(rf'&= \sqrt{{{Mcr_z:.2f} \
                            times {Mcr_T:.2f}}}\\'))
746                 mcr_eq.append(NoEscape(rf'&= {Mcr:.2f} \text{{ kN-m}}\\
                        '))
747                 mcr_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007,
                        Annex E]}\\'))
748                 mcr_eq.append(NoEscape(r'\end{aligned}'))
749
750                 report_check.append([
751                     NoEscape(r'Elastic Critical Moment'),
752                     '',
753                     mcr_eq,
754                     ''
755                 ])
756
757             # Lambda_LT
758             if lambda_LT != 'NA':
759                 lambda_eq = Math(inline=True)
760                 lambda_eq.append(NoEscape(r'\begin{aligned}\\'))
761                 lambda_eq.append(NoEscape(rf'\lambda_{{LT}} &= {
                        lambda_LT:.3f}\\'))
762                 lambda_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007,
                        Cl.8.2.2]}\\'))
763                 lambda_eq.append(NoEscape(r'\end{aligned}'))
764
765                 report_check.append([
766                     NoEscape(r'Slenderness Ratio'),
767                     '',
768                     lambda_eq,
769                     ''
770                 ])
771
772             # f_bd
```

```
            if fbd != 'NA' and chi_LT != 'NA':
                fbd_eq = Math(inline=True)
                fbd_eq.append(NoEscape(r'\begin{aligned}\\'))
                fbd_eq.append(NoEscape(r'f_{bd} &= \dfrac{\chi_{LT} \
                    times f_y}{\gamma_{m0}}\\'))
                fbd_eq.append(NoEscape(rf'&= \dfrac{{{chi_LT:.3f} \
                    times {fy}}}{{{gamma_m0}}}\\\\'))
                fbd_eq.append(NoEscape(rf'&= {fbd:.2f} \text{{ MPa
                    }}\\\\'))
                fbd_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                    .8.2.2]}\\'))
                fbd_eq.append(NoEscape(r'\end{aligned}'))

                report_check.append([
                    NoEscape(r'Design Bending Strength'),
                    '',
                    fbd_eq,
                    ''
                ])

        # ==================== DEFLECTION CHECK ====================
        report_check.append(['SubSection', 'Deflection Check',
            table_format])

        defl_val = getattr(pg_obj, 'deflection_criteria', 600)
        try:
             defl_limit_ratio = float(defl_val)
             if defl_limit_ratio <= 0:
                 defl_limit_ratio = 600
        except (ValueError, TypeError):
             defl_limit_ratio = 600

        if L > 0:

            # Allowable deflection
            delta_allow = round(L / defl_limit_ratio, 2)

            allow_eq = Math(inline=True)
            allow_eq.append(NoEscape(r'\begin{aligned}\\'))
```

```
807             allow_eq.append(NoEscape(rf'\delta_{{allow}} &= \dfrac{{L
                    }}{{{defl_limit_ratio}}}\\\\'))
808             allow_eq.append(NoEscape(rf'&= \dfrac{{{L:.1f}}}{{{
                    defl_limit_ratio}}}\\\\'))
809             allow_eq.append(NoEscape(rf'&= {delta_allow:.2f} \text{{ mm
                    }}\\\\'))
810             allow_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Table
                    6]}\\'))
811             allow_eq.append(NoEscape(r'\end{aligned}'))
812
813             report_check.append([
814                 NoEscape(r'Allowable Deflection'),
815                 '',
816                 allow_eq,
817                 ''
818             ])
819
820             # Actual deflection
821             delta_actual_str = getattr(pg_obj, 'calculated_deflection',
                    'NA')
822             if delta_actual_str not in ['NA', 'Skipped']:
823                 try:
824                     delta_actual = round(float(delta_actual_str), 2)
825
826                     # Calculate Iz for formula display
827                     try:
828                         Iz_mm4 = Unsymmetrical_I_Section_Properties.
                                calc_MomentOfAreaZ(
829                             D, bf_top, bf_bot, tw, tf_top, tf_bot
830                         )
831                     except:
832                         Iz_mm4 = 1
833
834                     # Calculate w (N/mm) from M_applied (kNm) assuming
                            UDL
835                     # M = w L^2 / 8 => w = 8 M / L^2
836                     if L > 0:
837                         M_Nmm = M_applied * 1e6
838                         w_udl = 8 * M_Nmm / (L**2)
```

```
                    else:
                        w_udl = 0

                    actual_eq = Math(inline=True)
                    actual_eq.append(NoEscape(r'\begin{aligned}\\'))
                    actual_eq.append(NoEscape(r'\delta &= \dfrac{5 w L
                        ^4}{384 E I_{z}}\\\\'))
                    actual_eq.append(NoEscape(rf'&= \dfrac{{5 \times {
                        w_udl:.2f} \times {L:.1f}^4}}{{384 \times {E} \
                        times {Iz_mm4:.2e}}}\\\\'))
                    actual_eq.append(NoEscape(rf'&= {delta_actual:.2f}
                        \text{{ mm}}\\\\'))
                    actual_eq.append(NoEscape(r'&\text{[Ref: IS
                        800:2007, Table 6]}'))
                    actual_eq.append(NoEscape(r'\end{aligned}'))

                    defl_status = 'Pass' if defl_check else 'Fail'
                    report_check.append([
                        'Actual Deflection',
                        NoEscape(rf'{delta_allow:.2f} \text{{ mm}}'),
                        actual_eq,
                        defl_status
                    ])
                except (ValueError, TypeError):
                    pass

        # ==================== WELD DESIGN ====================
        report_check.append(['SubSection', 'Weld Design', table_format
            ])

        weld_top = getattr(pg_obj, 'atop', 0)
        weld_bot = getattr(pg_obj, 'abot', 0)
        t_min = min(tw, tf_top, tf_bot) if tw > 0 and tf_top > 0 and
            tf_bot > 0 else 0
        weld_stiff = getattr(pg_obj, 'weld_stiff', None)

        # Minimum weld size per IS 800:2007 Table 21
        if t_min < 10:
            s_min = 3
```

```
        elif t_min <= 20:
            s_min = 5
        else:
            s_min = 6

        minweld_eq = Math(inline=True)
        minweld_eq.append(NoEscape(r'\begin{aligned}\\'))
        minweld_eq.append(NoEscape(rf'\text{{Thinner plate}} &= {t_min
            :.1f} \text{{ mm}}'))
        minweld_eq.append(NoEscape(r'\text{[Ref: IS 800:2007, Table
            21]}\\'))
        minweld_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'Minimum Weld Size',
            '',
            NoEscape(rf'{round(s_min, 1):.1f}\text{{ mm}}'),
            ''
        ])

        # Weld sizes - design outputs
        if weld_top > 0:
            wtop_eq = Math(inline=True)
            wtop_eq.append(NoEscape(r'\begin{aligned}\\'))
            wtop_eq.append(NoEscape(rf's_{{top}} &= {round(weld_top, 1)
                :.1f} \text{{ mm}}\\'))
            wtop_eq.append(NoEscape(r'\end{aligned}'))

            report_check.append([
                'Weld Size - Web to Top Flange',
                '',
                NoEscape(rf'{round(weld_top, 1):.1f} \text{{ mm}}'),
                ''
            ])

        if weld_bot > 0:
            wbot_eq = Math(inline=True)
            wbot_eq.append(NoEscape(r'\begin{aligned}\\'))
            wbot_eq.append(NoEscape(rf's_{{bot}} &= {round(weld_bot, 1)
```

```
            :.1f} \text{{ mm}}\\'))
        wbot_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'Weld Size - Web to Bottom Flange',
            '',
            NoEscape(rf'{round(weld_bot, 1):.1f} \text{{ mm}}'),
            ''
        ])

    if weld_stiff and weld_stiff not in [None, 'NA', 0, '']:
        weld_stiff_val = float(weld_stiff)
        if weld_stiff_val > 0:
            wstiff_eq = Math(inline=True)
            wstiff_eq.append(NoEscape(r'\begin{aligned}'))
            wstiff_eq.append(NoEscape(rf's_{{stiff}} &= {round(
                weld_stiff_val, 1):.1f} \text{{ mm}}'))
            wstiff_eq.append(NoEscape(r'\end{aligned}'))

            report_check.append([
                'Weld Size - Stiffener to Web',
                '',
                NoEscape(rf'{round(weld_stiff_val, 1):.1f} \text{{
                    mm}}'),
                ''
            ])



    # ==================== INTERMEDIATE STIFFENER - SECTION 1.5.1
        ====================
    report_check.append(['SubSection', 'Intermediate Stiffener',
        table_format])

    c = getattr(pg_obj, 'c', 0)

    if d > 0:
        c_d_ratio = round(c / d, 3)
```

```
            sqrt_2 = 1.414

            if c_d_ratio >= sqrt_2:
                I_s_min = round(0.75 * d * (tw**3), 2)

                I_s_eq = Math(inline=True)
                I_s_eq.append(NoEscape(r'\begin{aligned}\\'))
                I_s_eq.append(NoEscape(rf'\text{{Since }} \dfrac{{c}}{{
                    d}} &= {c_d_ratio:.3f} \geq \sqrt{{2}}\\\\'))
                I_s_eq.append(NoEscape(r'I_s &\geq 0.75 \, d \, t_w
                    ^3\\\\'))
                I_s_eq.append(NoEscape(rf'&\geq 0.75 \times {d:.1f} \
                    times ({tw:.1f})^3\\\\'))
                I_s_eq.append(NoEscape(rf'&\geq {I_s_min:.2f} \text{{
                    mm}}^4\\\\'))
                I_s_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                    .8.7.1.2, Eq. 1.17]}\\'))
                I_s_eq.append(NoEscape(r'\end{aligned}'))
            else:
                # condition_status = rf'Since \dfrac{{c}}{{d}} = {
                    c_d_ratio:.3f} < \sqrt{{2}}'
                I_s_min = round((1.5 * (d**3) * (tw**3)) / (c**2), 2)

                I_s_eq = Math(inline=True)
                I_s_eq.append(NoEscape(r'\begin{aligned}\\'))
                I_s_eq.append(NoEscape(rf'\text{{Since }} \dfrac{{c}}{{
                    d}} &= {c_d_ratio:.3f} < \sqrt{{2}}\\\\'))
                I_s_eq.append(NoEscape(r'I_s &\geq 1.5 \, \dfrac{d^3
                    t_w^3}{c^2}\\\\'))
                I_s_eq.append(NoEscape(rf'&\geq 1.5 \times \dfrac{{({d
                    :.1f})^3 \times ({tw:.1f})^3}}{{({c:.1f})^2}}\\\\'))
                I_s_eq.append(NoEscape(rf'&\geq {I_s_min:.2f} \text{{
                    mm}}^4\\\\'))
                I_s_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                    .8.7.1.2, Eq. 1.18]}\\'))
                I_s_eq.append(NoEscape(r'\end{aligned}'))

            report_check.append([
                'Minimum Moment of Inertia',
```

```
            '',
            I_s_eq,
            ''
        ])

        mu = 0.3
        tau_crc = round((kv * (3.14159**2) * E) / (12 * (1 - mu**2)
            * ((d/tw)**2)), 2)

        tau_crc_eq = Math(inline=True)
        tau_crc_eq.append(NoEscape(r'\begin{aligned}\\'))
        tau_crc_eq.append(NoEscape(r'\tau_{cr,e} &= \dfrac{K_v \pi
            ^2 E}{12(1-\mu^2)(d/t_w)^2}\\\\'))
        tau_crc_eq.append(NoEscape(rf'&= \dfrac{{{kv:.3f} \times \
            pi^2 \times {E}}}{{12(1-{mu}^2)({d:.1f}/{tw:.1f})
            ^2}}\\\\'))
        tau_crc_eq.append(NoEscape(rf'&= {tau_crc:.2f} \text{{ MPa
            }}\\\\'))
        tau_crc_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
            .8.4.2.2, Eq. 1.23]}\\'))
        tau_crc_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'Critical Buckling Stres',
            NoEscape(rf'$\mu = {mu}$'),
            tau_crc_eq,
            ''
        ])


    # ==================== END PANEL STIFFENER - SECTION 1.5.2
        ====================
    report_check.append(['SubSection', 'End Panel Stiffener',
        table_format])

    # Vertical Anchor Force
    Vp = round((d * tw * fy) / (3**0.5), 2)
    Vp_kN = round(Vp / 1000, 2)

```

```
        Vp_eq = Math(inline=True)
        Vp_eq.append(NoEscape(r'\begin{aligned}\\'))
        Vp_eq.append(NoEscape(r'V_p &= \dfrac{d \cdot t_w \cdot f_y}{\
            sqrt{3}}\\\\'))
        Vp_eq.append(NoEscape(rf'&= \dfrac{{{d:.1f} \times {tw:.1f} \
            times {fy:.1f}}}{{\sqrt{{3}}}}\\\\'))
        Vp_eq.append(NoEscape(rf'&= {Vp_kN:.2f} \text{{ kN}}\\\\'))
        Vp_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
            .8.4.2.2]}\\'))
        Vp_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'Vertical Anchor Force',
            '',
            Vp_eq,
            ''
        ])

        # Tension Flange Reaction
        Rtf = round(Vp_kN / 2, 2)

        Rtf_eq = Math(inline=True)
        Rtf_eq.append(NoEscape(r'\begin{aligned}\\'))
        Rtf_eq.append(NoEscape(r'R_{tf} &= \dfrac{V_p}{2}\\\\'))
        Rtf_eq.append(NoEscape(rf'&= \dfrac{{{Vp_kN:.2f}}}{{2}}\\\\'))
        Rtf_eq.append(NoEscape(rf'&= {Rtf:.2f} \text{{ kN}}\\\\'))
        Rtf_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
            .8.4.2.2]}\\'))
        Rtf_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'Tension Flange Reaction',
            '',
            Rtf_eq,
            ''
        ])

        # Tension Flange Moment
        Mtf = round(Vp_kN * d / 10, 2)
```

```
1035
1036            Mtf_eq = Math(inline=True)
1037            Mtf_eq.append(NoEscape(r'\begin{aligned}\\'))
1038            Mtf_eq.append(NoEscape(r'M_{tf} &= \dfrac{V_p \cdot d}{10}\\\\'
                    ))
1039            Mtf_eq.append(NoEscape(rf'&= \dfrac{{{Vp_kN:.2f} \times {d:.1f
                    }}}{{10}}\\\\'))
1040            Mtf_eq.append(NoEscape(rf'&= {Mtf:.2f} \text{{ kN-mm}}\\\\'))
1041            Mtf_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
                    .8.4.2.2]}\\'))
1042            Mtf_eq.append(NoEscape(r'\end{aligned}'))
1043
1044            report_check.append([
1045                'Tension Flange Moment',
1046                '',
1047                Mtf_eq,
1048                ''
1049            ])
1050
1051            if web_philosophy != "Thick Web without ITS":
1052                # Calculate end panel stiffener thickness based on tension
                        flange reaction
1053                Vp = (d * tw * fy) / math.sqrt(3)
1054                Rtf = Vp / 2  # Tension flange reaction
1055
1056                # Design stiffener for Rtf
1057                # Assume stiffener is a pair on both sides of web
1058                endstiffwidth = getattr(pg_obj, 'end_stiffwidth', 0)
1059                stiff_width = endstiffwidth
1060
1061                # Required thickness based on bearing stress
1062                #   _bearing   = Rtf / (2    stiff_width    t_stiff)     fy /
                          m0
1063                # Solving for t_stiff: t_stiff     (Rtf     m0 ) / (2
                        stiff_width    fy)
1064
1065                if stiff_width > 0:
1066                    t_req = (Rtf * gamma_m0) / (2 * stiff_width * fy)
1067
```

```
1068                # Use thickness from pg_obj if available (Optimization
                        source of truth)
1069                t_provided_obj = getattr(pg_obj, 'end_stiffthickness',
                        0)
1070
1071                if t_provided_obj and t_provided_obj > 0:
1072                    pg_obj.endstiffthickness = float(t_provided_obj)
1073                else:
1074                    # Fallback design logic if not provided
1075                    available_thk = [8, 10, 12, 16, 20, 25, 32, 40, 50]
1076
1077                    endstiffthickness = None
1078                    for thk in available_thk:
1079                        if float(thk) >= t_req:
1080                            pg_obj.endstiffthickness = float(thk)
1081                            break
1082
1083                    if pg_obj.endstiffthickness is None:
1084                        pg_obj.endstiffthickness = float(available_thk
                                [-1])  # Use maximum if all fail
1085                        logger.warning(f"End stiffener thickness {
                                pg_obj.endstiffthickness}mm may be
                                insufficient")
1086            else:
1087                pg_obj.endstiffthickness = 50.0
1088
1089            pg_obj.endpanelstiffenerthickness = pg_obj.
                    endstiffthickness
1090            logger.info(f"End Panel Stiffener Thickness: {pg_obj.
                    endpanelstiffenerthickness} mm")
1091        else:
1092            # Thick web without stiffeners
1093            pg_obj.endpanelstiffenerthickness = "NA"
1094            pg_obj.endstiffthickness = 0
1095
1096        # ==================== LONGITUDINAL STIFFENER - SECTION 1.5.3
                ====================
1097        report_check.append(['SubSection', 'Longitudinal Stiffener',
                table_format])
```

```
1098
1099        # Calculate epsilon and limits
1100        epsilon_w = round((250 / fy)**0.5, 3)
1101        limit_200_eps = round(200 * epsilon_w, 2)
1102        limit_250_eps = round(250 * epsilon_w, 2)
1103        limit_400_eps = round(400 * epsilon_w, 2)
1104
1105        d_tw_ratio = round(d / tw, 2)
1106
1107        # Row 1: Web Thickness Limits (Check for Longitudinal Stiffener
                Requirement)
1108        req_check_eq = Math(inline=True)
1109        req_check_eq.append(NoEscape(r'\begin{aligned}\\'))
1110        req_check_eq.append(NoEscape(rf'1.&\text{{Transverse Stiffeners
                only: }}\\'))
1111        req_check_eq.append(NoEscape((rf'&\dfrac{{d}}{{t_w}} \leq 200 \
               epsilon_w = {limit_200_eps:.2f}\\\\')))
1112        req_check_eq.append(NoEscape(rf'2.&\text{{With 1st Longitudinal
                Stiffener: }}\\'))
1113        req_check_eq.append(NoEscape(rf'&\dfrac{{d}}{{t_w}} \leq 250 \
               epsilon_w = {limit_250_eps:.2f}\\\\'))
1114        req_check_eq.append(NoEscape(rf'3.&\text{{With 2nd Longitudinal
                Stiffener: }}\\'))
1115        req_check_eq.append(NoEscape(rf'&\dfrac{{d}}{{t_w}} \leq 400 \
               epsilon_w = {limit_400_eps:.2f}\\\\'))
1116        req_check_eq.append(NoEscape(rf'4.&\text{{Actual Web
               Slenderness: }}\\\\'))
1117        req_check_eq.append(NoEscape(rf'&\dfrac{{d}}{{t_w}} = {
               d_tw_ratio:.2f}\\\\'))
1118
1119        long_stiff_required = False
1120        second_stiff_required = False
1121
1122        if d_tw_ratio <= limit_200_eps:
1123            req_check_eq.append(NoEscape(rf'&\text{{Since }} {
                   d_tw_ratio:.2f} \leq {limit_200_eps:.2f}, \text{{ Limit
                    Satsified.}}\\\\'))
1124            req_check_eq.append(NoEscape(r'&\text{Longitudinal
                   Stiffeners NOT Required.}\\'))
```

```
            req_check_eq.append(NoEscape(r'\end{aligned}'))

            report_check.append([
                'Web Thickness Limits',
                '',
                req_check_eq,
                ''
            ])

            # Add placeholder rows for consistency
            report_check.append(['First Stiffener Placement', '', 'Not
                Required', ''])
            report_check.append(['First Stiffener - Moment of Inertia'
                , '', 'Not Required', ''])
            report_check.append(['Second Stiffener (Neutral Axis)', ''
                , 'Not Required', ''])

        else:
            long_stiff_required = True
            req_check_eq.append(NoEscape(rf'&\text{{Since }} {
                d_tw_ratio:.2f} > {limit_200_eps:.2f}, \text{{ Limit
                NOT Satsified.}}\\\\'))
            req_check_eq.append(NoEscape(r'&\text{Longitudinal
                Stiffeners REQUIRED.}\\\\'))
            req_check_eq.append(NoEscape(r'\end{aligned}'))

            report_check.append([
                'Web Thickness Limits',
                '',
                req_check_eq,
                ''
            ])

            # First Stiffener Placement
            y_comp = round((D - tf_top - tf_bot) / 5, 2)

            y_comp_eq = Math(inline=True)
            y_comp_eq.append(NoEscape(r'\begin{aligned}\\'))
            y_comp_eq.append(NoEscape(r'y &= \dfrac{1}{5}(D - t_f -
```

```
            t_f)\\\\'))
        y_comp_eq.append(NoEscape(rf'&= \dfrac{{1}}{{5}}({D:.1f} -
            {tf_top:.1f} - {tf_bot:.1f})\\\\'))
        y_comp_eq.append(NoEscape(rf'&= {y_comp:.2f} \text{{ mm
            }}\\\\'))
        y_comp_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
            .8.7.1.3]}\\'))
        y_comp_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'First Stiffener Placement',
            'Distance from compression flange',
            y_comp_eq,
            ''
        ])

        # First Stiffener Design (Is_1 calculation)
        Is_1 = round(4 * c * (tw**3), 2)

        Is1_eq = Math(inline=True)
        Is1_eq.append(NoEscape(r'\begin{aligned}\\'))
        Is1_eq.append(NoEscape(r'I_s &\geq 4 c t_w^3\\\\'))
        Is1_eq.append(NoEscape(rf'&\geq 4 \times {c:.1f} \times ({
            tw:.1f})^3\\\\'))
        Is1_eq.append(NoEscape(rf'&\geq {Is_1:.2f} \text{{ mm
            }}^4\\\\'))
        Is1_eq.append(NoEscape(r'&\text{[Ref: IS 800:2007, Cl
            .8.7.2.4]}\\'))
        Is1_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'First Stiffener - Moment of Inertia',
            '',
            Is1_eq,
            ''
        ])

        # Check for Second Stiffener Requirement
        Is2_check_eq = Math(inline=True)
```

```
            Is2_check_eq.append(NoEscape(r'\begin{aligned}\\'))
            Is2_check_eq.append(NoEscape(r'\text{Limit for single
                longitudinal stiffener:}\\\\'))
            Is2_check_eq.append(NoEscape(rf'\dfrac{{d}}{{t_w}} \leq
                250 \epsilon_w = {limit_250_eps:.2f}\\\\'))
            Is2_check_eq.append(NoEscape(r'\text{Actual Web
                Slenderness:}\\\\'))
            Is2_check_eq.append(NoEscape(rf'\dfrac{{d}}{{t_w}} &= {
                d_tw_ratio:.2f}\\\\'))

            if d_tw_ratio > limit_250_eps:
                second_stiff_required = True
                Is2_check_eq.append(NoEscape(rf'\text{{Since }} {
                    d_tw_ratio:.2f} > {limit_250_eps:.2f}, \text{{
                    Limit NOT Satsified.}}\\\\'))
                Is2_check_eq.append(NoEscape(r'\text{Second Stiffener
                    at N.A. REQUIRED.}\\'))

                # Add 2nd stiffener calculation to the same block or
                    next
                d_2 = round(D - tf_top - tf_bot, 2)
                Is_2 = round(d_2 * (tw**3), 2)
                Is2_check_eq.append(NoEscape(r'I_s &\geq d_2 \times
                    t_w^3\\\\'))
                Is2_check_eq.append(NoEscape(rf'&\geq {d_2:.2f} \times
                     ({tw:.1f})^3\\\\'))
                Is2_check_eq.append(NoEscape(rf'&\geq {Is_2:.2f} \text
                    {{ mm}}^4\\\\'))
                Is2_check_eq.append(NoEscape(r'&\text{[Ref: IS
                    800:2007, Cl.8.7.13.2, Eq. 1.42]}\\'))
                Is2_check_eq.append(NoEscape(r'\end{aligned}'))

                report_check.append([
                    'Second Stiffener (Neutral Axis)',
                    '',
                    Is2_check_eq,
                    ''
                ])
            else:
```

```
1217                Is2_check_eq.append(NoEscape(rf'\text{{Since }} {
                        d_tw_ratio:.2f} \leq {limit_250_eps:.2f}, \text{{
                        Limit Satsified.}}\\\\'))
1218                Is2_check_eq.append(NoEscape(r'\text{NOT Required.}\\'
                        ))
1219                Is2_check_eq.append(NoEscape(r'\end{aligned}'))
1220
1221                report_check.append([
1222                    'Second Stiffener (Neutral Axis)',
1223                    '',
1224                    Is2_check_eq,
1225                    ''
1226                ])
1227
1228
1229        # ==================== STIFFENER DESIGN SUMMARY
                ====================
1230        report_check.append(['SubSection', 'Stiffener Design Summary',
                table_format])
1231
1232        # Get all stiffener parameters from pg_obj
1233        t_int_stiff = getattr(pg_obj, 'IntStiffThickness', 0)
1234        t_end_stiff = getattr(pg_obj, 'endstiffthickness', 0)
1235
1236        # Determine Longitudinal Stiffener values based on requirement
1237        if long_stiff_required:
1238            t_long_stiff = getattr(pg_obj, 'longstiffenerthk', 'NA')
1239            num_long = getattr(pg_obj, 'longstiffenerno', 'Not Required
                    ')
1240            stiff_1_pos = getattr(pg_obj, 'x1', 'Not Required')
1241            stiff_2_pos = getattr(pg_obj, 'x2', 'Not Required')
1242        else:
1243            t_long_stiff = 'Not Required'
1244            num_long = 'Not Required'
1245            stiff_1_pos = 'Not Required'
1246            stiff_2_pos = 'Not Required'
1247
1248        method_name = getattr(pg_obj, 'x', 'Simple Post Critical')
1249        int_spacing = getattr(pg_obj, 'c', 0)
```

```
1250
1251        # Calculate number of end panel stiffeners
1252        if isinstance(t_end_stiff, (int, float)) and t_end_stiff > 0:
1253            num_end = "2 (Pair)"
1254        else:
1255            num_end = "0"
1256
1257        summary_data = [
1258            ['Method', str(method_name)],
1259            ['End Panel Stiffener Thickness (mm)', f'{t_int_stiff:.1f}'
                if isinstance(t_int_stiff, (int, float)) else str(
                t_int_stiff)],
1260            ['Number of End Panel Stiffeners', num_end],
1261            ['Intermediate Stiffener Thickness (mm)', f'{t_int_stiff:.1
                f}' if isinstance(t_int_stiff, (int, float)) else str(
                t_int_stiff)],
1262            ['Intermediate Stiffener Spacing (mm)', f'{int_spacing:.1f}
                ' if isinstance(int_spacing, (int, float)) else str(
                int_spacing)],
1263            ['Longitudinal Stiffener Thickness (mm)', f'{t_long_stiff
                :.1f}' if isinstance(t_long_stiff, (int, float)) else
                str(t_long_stiff)],
1264            ['Number of Longitudinal Stiffeners', str(num_long)],
1265            ['Stiffener 1 Pos. from Comp. Flange (mm)', f'{stiff_1_pos
                :.2f}' if isinstance(stiff_1_pos, (int, float)) else str
                (stiff_1_pos)],
1266            ['Stiffener 2 Pos. from Comp. Flange (mm)', f'{stiff_2_pos
                :.2f}' if isinstance(stiff_2_pos, (int, float)) else str
                (stiff_2_pos)]
1267        ]
1268
1269        # Create formatted table
1270        for item in summary_data:
1271            param_name = item[0]
1272            param_value = item[1]
1273
1274            summary_eq = Math(inline=True)
1275            summary_eq.append(NoEscape(r'\begin{aligned}\\'))
1276            summary_eq.append(NoEscape(rf'\text{{{param_value}}}'))
```

```
            summary_eq.append(NoEscape(r'\end{aligned}'))

            report_check.append([
                param_name,
                '',
                summary_eq,
                ''
            ])


        # ==================== OVERALL DESIGN CHECK ====================
        report_check.append(['SubSection', 'Overall Design Check', table_format])

        # READ utilization ratios from pg_obj
        ur_moment = round(getattr(pg_obj, 'moment_ratio', 0), 3)
        ur_shear = round(getattr(pg_obj, 'shear_ratio', 0), 3)
        ur_deflection = round(getattr(pg_obj, 'deflection_ratio', 0), 3)

        # Moment utilization - show calculation
        moment_ur_eq = Math(inline=True)
        moment_ur_eq.append(NoEscape(r'\begin{aligned}\\'))
        moment_ur_eq.append(NoEscape(r'UR_M &= \dfrac{M_{applied}}{M_d}\\'))
        if Md > 0:
            moment_ur_eq.append(NoEscape(rf'&= \dfrac{{{M_applied:.2f}}}{{{Md:.2f}}}\\\\'))
        moment_ur_eq.append(NoEscape(rf'&= {ur_moment:.3f}\\\\'))
        moment_ur_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'Moment Utilization Ratio',
            moment_ur_eq,
            f'{ur_moment:.3f}',
            ''
        ])

```

```
        # Shear utilization - show calculation
        shear_ur_eq = Math(inline=True)
        shear_ur_eq.append(NoEscape(r'\begin{aligned}\\'))
        shear_ur_eq.append(NoEscape(r'UR_V &= \dfrac{V_{applied}}{V_d
            }\\\\'))
        if V_d > 0:
            shear_ur_eq.append(NoEscape(rf'&= \dfrac{{{V_applied:.2f
                }}}{{{V_d:.2f}}}\\\\'))
        shear_ur_eq.append(NoEscape(rf'&= {ur_shear:.3f}\\\\'))
        shear_ur_eq.append(NoEscape(r'\end{aligned}'))

        report_check.append([
            'Shear Utilization Ratio',
            shear_ur_eq,
            f'{ur_shear:.3f}',
            ''
        ])

        # Deflection utilization - show calculation if applicable
        if ur_deflection > 0:
            delta_actual_str = getattr(pg_obj, 'calculated_deflection',
                'NA')
            if delta_actual_str not in ['NA', 'Skipped']:
                try:
                    delta_actual = round(float(delta_actual_str), 2)
                    delta_allow = round(L / defl_limit_ratio, 2)

                    defl_ur_eq = Math(inline=True)
                    defl_ur_eq.append(NoEscape(r'\begin{aligned}\\'))
                    defl_ur_eq.append(NoEscape(r'UR_{\delta} &= \dfrac
                        {\delta_{actual}}{\delta_{allowable}}\\\\'))
                    if delta_allow > 0:
                        defl_ur_eq.append(NoEscape(rf'&= \dfrac{{{
                            delta_actual:.2f}}}{{{delta_allow:.2f}}}\\\\
                            '))
                    defl_ur_eq.append(NoEscape(rf'&= {ur_deflection:.3f
                        }\\\\'))
                    defl_ur_eq.append(NoEscape(r'\end{aligned}'))

```

```
                    report_check.append([
                        'Deflection Utilization Ratio',
                        defl_ur_eq,
                        f'{ur_deflection:.3f}',
                        ''
                    ])
                except (ValueError, TypeError):
                    pass

        # Overall design status - calculate and display
        overall_ur = max(ur_moment, ur_shear, ur_deflection)

        overall_eq = Math(inline=True)
        overall_eq.append(NoEscape(r'\begin{aligned}\\'))
        overall_eq.append(NoEscape(r'UR_{overall} &= \max(UR_M, UR_V, UR_{\delta})\\\\'))
        overall_eq.append(NoEscape(rf'&= \max({ur_moment:.3f}, {ur_shear:.3f}, {ur_deflection:.3f})\\\\'))
        overall_eq.append(NoEscape(rf'&= {round(overall_ur, 3):.3f}\\'))
        overall_eq.append(NoEscape(r'\end{aligned}'))

        final_status = 'Pass' if design_status else 'Fail'

        ur_text = "< 1" if overall_ur < 1 else "> 1"
        report_check.append([
            'Overall Design Status',
            overall_eq,
            NoEscape(rf'{round(overall_ur, 3):.3f}\text{{ {ur_text}}}'),
            final_status
        ])

        logger.info("DDCL-compliant design checks prepared successfully")

    except Exception as e:
        logger.error(f"Error preparing design checks: {str(e)}")
        import traceback
```

```
1377            logger.error(f"Traceback: {traceback.format_exc()}")
1378
1379        return report_check
1380    %-------------------- end code ---------------
```

2.3.1 Explanation of the Code

The `save_design` logic in `latex_report.py` is structured to handle the complexity of Plate Girder design, which includes multiple branching paths (Supported vs. Unsupported, Thick vs. Thin Web). Its main steps are:

1. **Data Extraction (`prepare_report_input`):** A comprehensive dictionary is built containing all user inputs and derived section properties (Mass, Area, I_z, I_y, etc.). This ensures the report header and "Section Details" table are populated with accurate, rounded values.

2. **Dynamic Check Generation** (`prepare_design_checks`): Unlike simpler modules, the checks here are highly conditional:

   - **Shear Path:** The code checks `web_philosophy` (Thick vs. Thin). If "Thin Web", it further branches to display "Simple Post Critical" or "Tension Field" calculations, showing specific formulas for τ_b, λ_w, or tension field angle ϕ and width w_{tf}.
   - **Moment Path:** It distinguishes between Laterally Supported (Yielding governed) and Unsupported (Buckling governed). For the latter, it explicitly writes out the M_{cr}, λ_{LT}, χ_{LT}, and f_{bd} calculation steps using `pylatex` Math objects.
   - **Stiffener Detailing:** It conditionally adds checks for Stiffener Spacing (c/d ratios), Minimum Inertia ($I_s \geq 1.5d^3t^3/c^2$), and Longitudinal Stiffener placement/inertia if required by the aspect ratio d/t_w.

3. **Formatting LaTeX Equations:** Complex formulas such as the Tension Field strength are constructed dynamically. The code substitutes the calculated values (e.g., substituting ϕ=34.5) into the LaTeX string before rendering, providing a "transparent" calculation report.

4. **Overall Utilization and Status:** The final utilization ratio is derived as the maximum of Shear, Moment, and Deflection ratios. A final "PASS/FAIL" status is determined, and the appropriate summary (Safe/Unsafe) is flagged for the PDF generation.

5. **File Handling and PDF Generation:** Finally, the function sets up the output directory (creating it if required), and calls the common `CreateLatex.save_latex` routine. This compiles the LaTeX file into a professional PDF report that includes input summary, detailed calculation steps, design check table and conclusion for the welded butt joint.

2.4 Documentation

The directory structure used for the Report Generation is organized as follows:

```
src/osdag_core

 design_report/
    reportGenerator_latex.py

 design_type/
    plate_girder/
       report/
   |       latex_report.py
Report_functions.py
```

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

```
$ python -m osdag_gui
```

2.5 References

- Osdag Github repository: https://osdag.github.io/

- IS 800:2007, "General Construction in Steel Code of Practice of Practice"

Chapter 3

Report Generation of Bolted Lap Joint

3.1 Problem Statement

The primary objective of this task was to generate a comprehensive design report for **Bolted Lap Joint Connections**. Osdag (Open Steel Design and Graphics) is an open-source tool designed for the design and detailing of steel structures in accordance with Indian Standards. A critical requirement for steel structures is ensuring they pass all Design and Detailing Checklists (DDCL) to guarantee durability, and such comprehensive design reports provide this essential information. The challenge involved meticulously converting all necessary steps, formulas, and compliance checks stipulated by IS 800:2007 into code to ensure successful report generation for bolted connections.

3.2 Tasks Done

3.2.1 Core Save Design Function Development

- **Implemented comprehensive `save_design()` function in `lap_joint_bolted.py` with robust error handling and validation**

  Developed the main orchestrating function that serves as the entry point for

report generation. This function coordinates data collection from the UI, validates all design parameters (such as plate thickness, bolt grade, and load values), performs calculations, and triggers the LaTeX report compilation. It includes comprehensive error handling to manage exceptions during file operations and data processing.

– **Integrated with existing Osdag LaTeX report infrastructure using `CreateLatex` class**

  Successfully interfaced the new bolted lap joint module with Osdag's established LaTeX report generation framework. This required constructing a detailed `report_input` dictionary and a structured `report_check` list to pass data dynamically to the `CreateLatex.save_latex` method, ensuring consistency with other connection modules.

– **Implemented proper file path handling with OS-independent path management**

  Coded cross-platform file path management using Python's `os.path` module to ensure the report generation works seamlessly across different operating systems. The function automatically handles directory creation for reports if they do not exist.

– **Report Input Dictionary Structure**

  Architected a complete data structure that captures every piece of information needed for report generation, including:

  * Module identification (Lap Joint Bolted Connection)
  * Material properties (Ultimate and Yield stress)
  * Geometric parameters (Plate thicknesses, width)
  * Bolt specifics (Diameter, Grade, Type - Friction/Bearing)
  * Load information (Tensile/Compression Force)

3.2.2 Detailed Design Verification and Check Implementation

Bolt Capacity Checks

– **Implemented Bolt Shear and Bearing calculations**

Coded the mathematical logic for determining bolt capacity based on IS 800:2007 requirements [Cl. 10.3.3, 10.3.4]. The function dynamically calculates capacity based on the bolt type (Bearing or Friction Grip).

Shear Capacity:

$$V_{dsb} = \frac{f_{ub} \times A_{sb}}{\gamma_{mb}} \tag{3.1}$$

Bearing Capacity:

$$V_{dpb} = \frac{2.5 \times k_b \times d \times t \times f_u}{\gamma_{mb}} \tag{3.2}$$

– **Bolt Design Capacity Logic**

Implemented logic to determine the governing bolt value (V_{db}) by taking the minimum of shear and bearing capacities for bearing bolts, or slip resistance for friction bolts.

Base Metal Strength Analysis

– **Implemented dual strength checks (Yielding and Rupture)**

Developed parallel calculation paths for both failure modes in tension member design [Cl. 6.2, 6.3]:

Gross Yielding:

$$T_{dg} = \frac{A_g \times f_y}{\gamma_{m0}} \tag{3.3}$$

Net Rupture:

$$T_{dn} = \frac{0.9 \times A_n \times f_u}{\gamma_{m1}} \tag{3.4}$$

– **Compression Capacity Check**

Added conditional logic to handle compression loads [Cl. 7.1.2], ensuring the

report adapts dynamically to the `design_for` parameter:

$$P_d = \frac{A_g \times f_y}{\gamma_{m0}} \tag{3.5}$$

3.2.3 Advanced LaTeX Mathematical Formatting

Mathematical Equation Integration

– **Used `NoEscape` LaTeX formatting for complex mathematical expressions**

Mastered the `pylatex.utils.NoEscape` functionality to render complex mathematical equations without Python string escaping issues. This was crucial for rendering fractions, subscripts, and Greek letters (e.g., γ_{mb}, π) correctly in the final PDF.

– **Dynamic Equation Generation**

Implemented formatted strings (f-strings) within the LaTeX code to inject calculated Python variables (like `bolt_shear_kN`, `kb`, `An`) directly into the equation display strings. This ensures the report shows the actual numerical substitution steps, not just the final result.

3.2.4 Design Calculations Integration

Utilization Ratio and Detailing

– **Calculated Utilization Ratio (UR)**

Implemented a tracking system to calculate the utilization ratio for both the bolts and the base metal. The code identifies the critical UR to determine the overall safety of the connection:

$$UR = \frac{\text{Applied Force}}{\text{Design Capacity}} \tag{3.6}$$

– **Detailing Requirements Implementation**

Implemented practical construction detailing checks as per IS 800:2007 [Cl. 10.2], verifying:

* Minimum Pitch and Gauge ($2.5 \times d$)
* Minimum and Maximum Edge Distances

The `save_design` function automatically verifies if provided spacing (p_{prov}, e_{prov}) meets the calculated limits (p_{min}, e_{min}) and reports a "PASS" or "FAIL" status in the final document.

3.3 Python Code

The entire logic for the report generation of Welded Butt Joint module is within the save_design() function. And it's being connected to the main GUI through ui_template.py and ui_popup_summary.py files. Below are key snippets from the module with explanations of their functionality.

Description of the Script

The script defines the save_design() function, which handles everything from user input to final design report generation. It uses reportGenerator_latex.py to manage properties and perform calculations.

Python Code

The following snippets showcase the core logic implemented in the report generation task.

Listing 3.1: Bolted Lap Joint save_design() function

```
1 %--------------------begin code-------------
2 def save_design(self, popup_summary):
3         """
4         Generate the LaTeX design report for Lap Joint Bolted
             Connection (Tension/Compression)
5         per IS 800:2007.
6         """
7         try:
8             def g(attr, default=None):
9                 v = getattr(self, attr, default)
```

```
                return default if v is None else v

            def f2(x, default=0.0):
                try:
                    return round(float(x), 2)
                except (TypeError, ValueError):
                    return default

            def as_int(x, default=0):
                try:
                    return int(round(float(x)))
                except (TypeError, ValueError):
                    return default

            if not getattr(self, 'design_status', False):
                self.report_input = {
                    KEY_MODULE: "Lap Joint Bolted Connection",
                    KEY_MAIN_MODULE: "Plate(d) Connection",
                    "Design Status": "TITLE",
                    "Status": "Design not completed successfully.",
                }
                self.report_check = []
                self.report_check.append([
                    "SubSection", "Design Status", "|p{4cm}|p{5cm}|
                       p{5.5cm}|p{1.5cm}|"
                ])
                self.report_check.append(["Design", "Design not
                   completed successfully.", "", "FAIL"])

                Disp_2d_image = []
                Disp_3D_image = "/ResourceFiles/images/3d.png"
                rel_path = os.path.abspath(".").replace("\\", "/")
                fname_no_ext = popup_summary.get("filename", "
                   LapJointBoltedReport")
                folder = popup_summary.get('folder', './reports')
                os.makedirs(folder, exist_ok=True)

                CreateLatex.save_latex(
                    CreateLatex(), self.report_input, self.
```

```
                        report_check,
46                  popup_summary, fname_no_ext, rel_path,
                        Disp_2d_image, Disp_3D_image,
47                  module=getattr(self, 'module', 'Lap Joint
                        Bolted')
48              )
49              return True
50
51          self.module = g('module', 'Lap Joint Bolted')
52          self.mainmodule = 'Plate(d) Connection'
53          design_for = str(g('design_for', 'Tension')).strip()
54          is_comp = design_for.lower().startswith('c')
55
56          plate1_thk = f2(g('plate1thk', g('platethk', 0.0)), 0.0)
57          plate2_thk = f2(g('plate2thk', g('platethk', 0.0)), 0.0)
58          width = f2(g('width', 0.0), 0.0)
59          axial_kN = f2(g('axial_force_kN', g('tensile_force',
                0.0)), 0.0)
60
61          edge_type = getattr(self.bolt, 'edgetype', 'Sheared or
                hand flame cut')
62          bolt_dia_prov = f2(getattr(self.bolt, '
                bolt_diameter_provided', 0.0) if hasattr(self, 'bolt
                ') else 0.0, 0.0)
63          bolt_grade_prov = f2(getattr(self.bolt, '
                bolt_grade_provided', 0.0) if hasattr(self, 'bolt')
                else 0.0, 0.0)
64          bolt_type = getattr(self.bolt, 'bolt_type',
                VALUE_NOT_APPLICABLE) if hasattr(self, 'bolt') else
                VALUE_NOT_APPLICABLE
65
66          bolt_shear_kN = f2(getattr(self.bolt, '
                bolt_shear_capacity', 0.0) if hasattr(self, 'bolt')
                else 0.0, 0.0)
67          bolt_bearing_kN = f2(getattr(self.bolt, '
                bolt_bearing_capacity', 0.0) if hasattr(self, 'bolt'
                ) else 0.0, 0.0)
68          bolt_final_cap = f2(getattr(self.bolt, 'bolt_capacity',
                 0.0) if hasattr(self, 'bolt') else 0.0, 0.0)
```

```
69
70              rows = as_int(g('rows', 0), 0)
71              cols = as_int(g('cols', 0), 0)
72              n_bolts = as_int(g('number_bolts', 0), 0)
73              pitch = f2(g('final_pitch', 0.0), 0.0)
74              gauge = f2(g('final_gauge', 0.0), 0.0)
75              e_dist = f2(g('final_edge_dist', 0.0), 0.0)
76
77              t_fu_fy_list = getattr(self, 'bolt_conn_plates_t_fu_fy'
                    , [])
78              if t_fu_fy_list and len(t_fu_fy_list) > 0:
79                  fu = t_fu_fy_list[0][1] if len(t_fu_fy_list[0]) > 1
                        else 0
80                  fy = t_fu_fy_list[0][2] if len(t_fu_fy_list[0]) > 2
                        else 0
81              else:
82                  fy = g('yield_stress', 0)
83                  fu = 0
84
85              base_metal_capacity_kN = f2(g('base_metal_capacity_kN',
                    0.0), 0.0)
86
87              A_g = f2(g('A_g', 0.0), 0.0)
88              T_dg = f2(g('T_dg', 0.0), 0.0)
89              T_dn = f2(g('T_dn', 0.0), 0.0)
90              T_db = f2(g('T_db', 0.0), 0.0)
91
92              overall_ur = round(g('utilization_ratio', 0.0), 3)
93
94              self.report_input = {
95                  KEY_MODULE: "Lap Joint Bolted Connection",
96                  KEY_MAIN_MODULE: "Plate(d) Connection",
97                  KEY_DISP_DESIGN_FOR: design_for,
98                  "Thickness of Plate-1 (mm) *": plate1_thk,
99                  "Thickness of Plate-2 (mm) *": plate2_thk,
100                 "Width of Plate (mm) *": width,
101                 "Material *": getattr(self, 'main_material',
                        VALUE_NOT_APPLICABLE),
102                 "Diameter (mm) *": bolt_dia_prov,
```

```
103                 "Property Class *": bolt_grade_prov,
104                 "Type *": bolt_type,
105                 f"{'Tensile' if not is_comp else 'Axial'} Force (kN
                        ) *": axial_kN,
106                 "Additional inputs": "TITLE",
107                 "Bolt Hole Type": getattr(self.bolt, 'boltholetype'
                        , 'Standard'),
108                 "Slip Factor ( f )": getattr(self.bolt, 'mu_f', 'N/
                        A'),
109                 "Edge Preparation Method": edge_type
110             }
111
112             self.report_check = []
113
114             #
                    ============================================================

115             #=========== SECTION 2.1: CALCULATING BOLT STRENGTH
                    ==========
116             #
                    ============================================================

117             self.report_check.append([
118                 "SubSection", "Calculating Bolt Strength", "|p{4cm
                        }|p{5cm}|p{5.5cm}|p{1.5cm}|"
119             ])
120
121             d = float(self.bolt.bolt_diameter_provided)
122             bolt_grade = float(self.bolt.bolt_grade_provided)
123             f_ub = int(bolt_grade * 100)
124
125             plate1_thk_raw = float(self.plate1.thickness[0]) if
                    isinstance(self.plate1.thickness, list) else float(
                    self.plate1.thickness)
126             plate2_thk_raw = float(self.plate2.thickness[0]) if
                    isinstance(self.plate2.thickness, list) else float(
                    self.plate2.thickness)
127
128             bolt_shank_area = f2(math.pi * d**2 / 4, 0.0)
```

```
129
130             if hasattr(self.bolt, 'bolt_net_area_provided'):
131                 bolt_net_area = f2(self.bolt.bolt_net_area_provided
                        , 0.0)
132             else:
133                 bolt_net_area = f2(math.pi * (d - 0.9382 * math.
                        sqrt(d))**2 / 4, 0.0)
134
135             gamma_mb = 1.25
136
137             if self.bolt.bolt_type != "Bearing Bolt":
138                 # ========== FRICTION GRIP TYPE BOLTING (Cl.
                        10.4.3) ==========
139                 f_0 = 0.7 * f_ub
140                 F_o = bolt_net_area * f_0
141                 mu = float(self.bolt.mu_f) if hasattr(self.bolt, '
                        mu_f') else 0.3
142                 n_e = 1  # Number of effective interfaces (single
                        lap joint)
143
144                 bolt_hole_type_str = str(self.bolt.bolt_hole_type)
                        if hasattr(self.bolt, 'bolt_hole_type') else "
                        Standard"
145                 d_0 = IS800_2007.cl_10_2_1_bolt_hole_size(d,
                        bolt_hole_type_str)
146
147                 hole_type_lower = bolt_hole_type_str.lower()
148                 if "standard" in hole_type_lower:
149                     K_h = 1.0
150                 elif "over" in hole_type_lower or "short" in
                        hole_type_lower:
151                     K_h = 0.85
152                 else:  # long slotted
153                     K_h = 0.7
154
155                 V_nsf = mu * n_e * K_h * F_o
156                 gamma_mf = 1.25  # For ultimate load
157                 V_dsf_theoretical = V_nsf / gamma_mf
158                 V_dsf_kN_theoretical = V_dsf_theoretical / 1000
```

```
159
160                 slip_req = Math(inline=True)
161                 slip_req.append(NoEscape(r'\begin{aligned}'))
162                 slip_req.append(NoEscape(r'V_{dsf} &= \frac{V_{nsf
                        }}{\gamma_{mf}}\\'))
163                 slip_req.append(NoEscape(r'V_{nsf} &= \mu \cdot n_e
                        \cdot K_h \cdot F_o\\'))
164                 slip_req.append(NoEscape(r'\mu &= ' + f'{mu:.2f}' +
                        r' \text{ (slip factor)}\\'))
165                 slip_req.append(NoEscape(r'n_e &= ' + str(n_e) + r'
                        \text{ (interfaces)}\\'))
166                 slip_req.append(NoEscape(r'K_h &= ' + f'{K_h:.2f}'
                        + r' \text{ (hole factor)}\\'))
167                 slip_req.append(NoEscape(r'f_0 &= 0.7 f_{ub} = ' +
                        f'{f_0:.1f}' + r' \text{ MPa}\\'))
168                 slip_req.append(NoEscape(r'A_{nb} &= ' + f'{
                        bolt_net_area:.2f}' + r' \text{ mm}^2\\'))
169                 slip_req.append(NoEscape(r'F_o &= A_{nb} \times f_0
                        = ' + f'{F_o:.2f}' + r' \text{ N}\\'))
170                 slip_req.append(NoEscape(r'V_{nsf} &= ' + f'{mu:.2f
                        }' + r' \times ' + str(n_e) + r' \times ' + f'{
                        K_h:.2f}' + r' \times ' + f'{F_o:.2f}' + r'\\'))
171                 slip_req.append(NoEscape(r'&= ' + f'{V_nsf:.2f}' +
                        r' \text{ N}\\'))
172                 slip_req.append(NoEscape(r'V_{dsf} &= \frac{' + f'{
                        V_nsf:.2f}' + r'}{' + str(gamma_mf) + r'} = ' +
                        f'{V_dsf_kN_theoretical:.2f}' + r' \text{ kN}\\'
                        ))
173                 slip_req.append(NoEscape(r'&[\text{Ref. Cl.
                        10.4.3}]'))
174                 slip_req.append(NoEscape(r'\end{aligned}'))
175
176                 self.report_check.append(["Slip Resistance", "",
                        slip_req, ""])
177
178             else:  # Bearing Bolt
179                 # ========== SHEAR CAPACITY (Cl. 10.3.3) ==========
180                 V_dsb_kN = bolt_shear_kN
181                 V_nsb = V_dsb_kN * gamma_mb
```

```
182
183                 n_n = 1  # Threads intercepting shear plane
184                 n_s = 0  # No threads without shear
185
186                 shear_req = Math(inline=True)
187                 shear_req.append(NoEscape(r'\begin{aligned}\\'))
188                 shear_req.append(NoEscape(r'V_{dsb} &= \frac{V_{nsb
                        }}{\gamma_{mb}}\\\\'))
189                 shear_req.append(NoEscape(r'V_{nsb} &= \frac{f_{ub
                        }}{\sqrt{3}} \cdot (n_n \cdot A_{nb} + n_s \cdot
                         A_{sb})\\'))
190                 shear_req.append(NoEscape(r'&= \frac{' + str(f_ub)
                        + r'}{\sqrt{3}} \times (1 \times ' + f'{
                        bolt_net_area:.2f}' + r')\\'))
191                 shear_req.append(NoEscape(r'&= ' + f'{V_nsb:.2f}' +
                         r' \text{ kN}\\\\'))
192                 shear_req.append(NoEscape(r'V_{dsb} &= \frac{' + f'
                        {V_nsb:.2f}' + r'}{' + str(gamma_mb) + r'}\\'))
193                 shear_req.append(NoEscape(r'&= ' + f'{V_dsb_kN:.2f}
                        ' + r' \text{ kN}\\'))
194                 shear_req.append(NoEscape(r'&[\text{Ref. Cl.
                        10.3.3}]'))
195                 shear_req.append(NoEscape(r'\end{aligned}'))
196
197                 self.report_check.append(["Shear Capacity", "",
                        shear_req, ""])
198
199                 # ========== BEARING CAPACITY (Cl. 10.3.4)
                        ==========
200                 V_dpb_kN = bolt_bearing_kN
201                 V_npb = V_dpb_kN * gamma_mb
202
203                 t_min = min(plate1_thk_raw, plate2_thk_raw)
204                 f_u_plate = min(self.plate1.fu, self.plate2.fu)
205
206                 bolt_hole_type_str = str(self.bolt.bolt_hole_type)
                        if hasattr(self.bolt, 'bolt_hole_type') else "
                        Standard"
207                 d_0 = IS800_2007.cl_10_2_1_bolt_hole_size(d,
```

```
                    bolt_hole_type_str)
208
209                 e = float(self.final_end_dist) if hasattr(self, '
                        final_end_dist') and self.final_end_dist > 0
                        else float(self.bolt.min_end_dist_round)
210                 p = float(self.final_pitch) if hasattr(self, '
                        final_pitch') and self.final_pitch > 0 else
                        float(self.bolt.min_pitch_round)
211
212                 # Calculate kb factor components (always calculate
                        for report display)
213                 if p > 0:
214                     kb_1 = e / (3.0 * d_0)
215                     kb_2 = p / (3.0 * d_0) - 0.25
216                     kb_3 = f_ub / f_u_plate
217                     kb_4 = 1.0
218                     kb_calc = min(kb_1, kb_2, kb_3, kb_4)
219                 else:
220                     kb_1 = e / (3.0 * d_0)
221                     kb_2 = float('inf')  # Not applicable
222                     kb_3 = f_ub / f_u_plate
223                     kb_4 = 1.0
224                     kb_calc = min(kb_1, kb_3, kb_4)
225
226                 if hasattr(self.bolt, 'kb') and self.bolt.kb is not
                        None:
227                     k_b = f2(self.bolt.kb, 1.0)
228                 else:
229                     k_b = f2(kb_calc, 1.0)
230
231                 kb_req = Math(inline=True)
232                 kb_req.append(NoEscape(r'\begin{aligned}\\'))
233                 kb_req.append(NoEscape(r'k_b &= \min\left(\frac{e
                        }{3d_0}, \frac{p}{3d_0}-0.25, \frac{f_{ub}}{f_u
                        }, 1.0\right)\\\\'))
234                 kb_req.append(NoEscape(r'&= \min\left(\frac{' + f'{
                        e:.1f}' + r'}{3 \times ' + f'{d_0:.1f}' + r'}, \
                        frac{' + f'{p:.1f}' + r'}{3 \times ' + f'{d_0:.1
                        f}' + r'}-0.25, \frac{' + str(f_ub) + r'}{' +
```

```
                    str(f_u_plate) + r'}, 1.0\right)\\\\'))
235
236                 if p > 0:
237                     kb_req.append(NoEscape(r'&= \min(' + f'{kb_1:.2
                            f}' + r', ' + f'{kb_2:.2f}' + r', ' + f'{
                            kb_3:.2f}' + r', 1.0)\\\\'))
238                 else:
239                     kb_req.append(NoEscape(r'&= \min(' + f'{kb_1:.2
                            f}' + r', ' + f'{kb_3:.2f}' + r', 1.0)\\\\')
                            )
240
241                 kb_req.append(NoEscape(r'&= ' + f'{k_b:.2f}'))
242                 kb_req.append(NoEscape(r'\end{aligned}'))
243                 self.report_check.append(["Bearing Factor", "",
                        kb_req, ""])
244
245                 bearing_req = Math(inline=True)
246                 bearing_req.append(NoEscape(r'\begin{aligned}'))
247                 bearing_req.append(NoEscape(r'V_{dpb} &= \frac{V_{
                        npb}}{\gamma_{mb}}\\\\'))
248                 bearing_req.append(NoEscape(r'V_{npb} &= 2.5 \cdot
                        k_b \cdot d \cdot t \cdot f_u\\'))
249                 bearing_req.append(NoEscape(r'&= 2.5 \times ' + f'{
                        k_b:.3f}' + r' \times ' + f'{d:.1f}' + r' \times
                         ' + f'{t_min:.1f}' + r' \times ' + str(
                        f_u_plate) + r'\\'))
250                 bearing_req.append(NoEscape(r'&= ' + f'{V_npb:.2f}'
                         + r' \text{ kN}\\\\'))
251                 bearing_req.append(NoEscape(r'V_{dpb} &= \frac{' +
                        f'{V_npb:.2f}' + r'}{' + f'{gamma_mb:.2f}' + r'
                        }\\'))
252                 bearing_req.append(NoEscape(r'&= ' + f'{V_dpb_kN:.2
                        f}' + r' \text{ kN}\\'))
253                 bearing_req.append(NoEscape(r'&[\text{Ref. Cl.
                        10.3.4}]'))
254                 bearing_req.append(NoEscape(r'\end{aligned}'))
255
256                 self.report_check.append(["Bearing Capacity", "",
                        bearing_req, ""])
```

```
257
258                 V_db_kN = bolt_final_cap
259
260                 cap_req = Math(inline=True)
261                 cap_req.append(NoEscape(r'\begin{aligned}'))
262                 cap_req.append(NoEscape(r'V_{db} &= \min(' + f'{
                        bolt_shear_kN:.2f}' + r', ' + f'{bolt_bearing_kN
                        :.2f}' + r')'+r'\\'))
263                 cap_req.append(NoEscape(r'&= ' + f'{V_db_kN:.2f}' +
                         r' \text{ kN}'+r'\\'))
264                 cap_req.append(NoEscape(r'&[\text{Ref. Cl. 10.3.2,
                        IS 800:2007}]'))
265                 cap_req.append(NoEscape(r'\end{aligned}'))
266
267                 self.report_check.append(["Bolt Design Capacity", "
                        ", cap_req, ""])
268
269             #
                    ============================================================
270             #=========== SECTION 2.2: REDUCTION FACTORS
                    ============
271             #
                    ============================================================
272             self.report_check.append([
273                 "SubSection", "Reduction Factors", "|p{4cm}|p{5cm}|
                        p{5.5cm}|p{1.5cm}|"
274             ])
275
276             l_j = (self.rows - 1) * self.final_pitch if self.rows >
                    1 else 0
277             d = self.bolt.bolt_diameter_provided
278
279             lj_req = Math(inline=True)
280             lj_req.append(NoEscape(r'\begin{aligned}'))
281
282             if l_j > 15 * d:
283                 beta_lj = 1.075 - (l_j / (200 * d))
```

```
                beta_lj = max(0.75, min(beta_lj, 1.0))
                lj_req.append(NoEscape(r'\text{Since } l_j &> 15d\\'))
                lj_req.append(NoEscape(r'l_j &= ' + str(l_j) + r' \text{ mm}, \quad 15d = ' + str(15 * d) + r' \text{ mm}\\'))
                lj_req.append(NoEscape(r'\beta_{lj} &= 1.075 - \frac{l_j}{200 \cdot d}\\'))
                lj_req.append(NoEscape(r'&= 1.075 - \frac{' + str(l_j) + r'}{200 \times ' + str(d) + r'}\\'))
                lj_req.append(NoEscape(r'&= ' + f'{1.075 - (l_j / (200 * d)):.3f}' + r'\\'))
                lj_req.append(NoEscape(r'&\text{(but } 0.75 \leq \beta_{lj} \leq 1.0\text{)}\\'))
                lj_req.append(NoEscape(r'\beta_{lj} &= ' + f'{beta_lj:.2f}' + r'\\'))
                lj_status = ""
            else:
                beta_lj = 1.0
                lj_req.append(NoEscape(r'\text{Since } l_j &\leq 15d\\'))
                lj_req.append(NoEscape(r'l_j &= ' + str(l_j) + r' \text{ mm}, \quad 15d = ' + str(15 * d) + r' \text{ mm}\\'))
                lj_req.append(NoEscape(r'\beta_{lj} &= 1.0 \text{ (No reduction)}\\'))
                lj_status = "PASS"

            lj_req.append(NoEscape(r'&[\text{Ref. Cl. 10.3.3.1}]'))
            lj_req.append(NoEscape(r'\end{aligned}'))

            lj_prov = Math(inline=True)
            lj_prov.append(NoEscape(r'\beta_{lj} = ' + f'{beta_lj:.2f}'))

            self.report_check.append(["Long Joint Factor", lj_req, lj_prov, ''])

            l_g = plate1_thk_raw + plate2_thk_raw
```

```
309
310             lg_req = Math(inline=True)
311             lg_req.append(NoEscape(r'\begin{aligned}'))
312
313             if l_g > 5 * d:
314                 beta_lg = (8 * d) / (3 * d + l_g)
315                 beta_lg = min(beta_lg, beta_lj) if beta_lj else
                        beta_lg
316                 lg_req.append(NoEscape(r'\text{Since } l_g &> 5d\\'
                        ))
317                 lg_req.append(NoEscape(r'l_g &= ' + str(l_g) + r' \
                        text{ mm}, \quad 5d = ' + str(5 * d) + r' \text{
                         mm}\\'))
318                 lg_req.append(NoEscape(r'\beta_{lg} &= \frac{8d}{3d
                         + l_g}\\'))
319                 lg_req.append(NoEscape(r'&= \frac{8 \times ' + str(
                        d) + r'}{3 \times ' + str(d) + r' + ' + str(l_g)
                         + r'}\\'))
320                 lg_req.append(NoEscape(r'&= ' + f'{(8 * d) / (3 * d
                         + l_g):.3f}' + r'\\'))
321                 lg_req.append(NoEscape(r'\beta_{lg} &\leq \beta_{lj
                        }\\'))
322                 lg_req.append(NoEscape(r'\beta_{lg} &= ' + f'{
                        beta_lg:.2f}' + r'\\'))
323                 lg_status = ""
324             else:
325                 beta_lg = 1.0
326                 lg_req.append(NoEscape(r'\text{Since } l_g &\leq 5d
                        \\'))
327                 lg_req.append(NoEscape(r'l_g &= ' + str(l_g) + r' \
                        text{ mm}, \quad 5d = ' + str(5 * d) + r' \text{
                         mm}\\'))
328                 lg_req.append(NoEscape(r'\beta_{lg} &= 1.0 \text{ (
                        No reduction)}\\'))
329                 lg_status = "PASS"
330
331             lg_req.append(NoEscape(r'&[\text{Ref. Cl. 10.3.3.2}]'))
332             lg_req.append(NoEscape(r'\end{aligned}'))
333
```

```
334             lg_prov = Math(inline=True)
335             lg_prov.append(NoEscape(r'\beta_{lg} = ' + f'{beta_lg
                    :.2f}'))
336 
337             self.report_check.append(["Large Grip Factor", lg_req,
                    lg_prov, ''])
338 
339             if self.bolt.bolt_hole_type != "Standard":
340                 hole_req = Math(inline=True)
341                 hole_req.append(NoEscape(r'\begin{aligned}'))
342                 hole_req.append(NoEscape(r'\text{Hole Type: }' +
                        self.bolt.bolt_hole_type + r'\\'))
343                 if "oversized" in self.bolt.bolt_hole_type.lower()
                        or "short" in self.bolt.bolt_hole_type.lower():
344                     hole_factor = 0.7
345                 else:  # long-slotted
346                     hole_factor = 0.5
347                 hole_req.append(NoEscape(r'\text{Reduction Factor}
                        &= ' + str(hole_factor) + r'\\'))
348                 hole_req.append(NoEscape(r'&[\text{Ref. Cl.
                        10.3.4}]'))
349                 hole_req.append(NoEscape(r'\end{aligned}'))
350                 self.report_check.append(["Hole Type Reduction",
                        hole_req, "", ""])
351 
352             #====================================
353             # Section 2.3: Detailing Requirements
354             #====================================
355             self.report_check.append([
356                 "SubSection", "Detailing Requirements", "|p{4cm}|p
                        {5cm}|p{5.5cm}|p{1.5cm}|"
357             ])
358 
359             # 2.3.1 Minimum Spacing (Cl. 10.2.2)
360             p_min = as_int(2.5 * bolt_dia_prov, 0)
361             g_min = as_int(2.5 * bolt_dia_prov, 0)
362 
363             spacing_req = Math(inline=True)
364             spacing_req.append(NoEscape(r'\begin{aligned}'))
```

```
365             spacing_req.append(NoEscape(r'p_{\text{min}} &= 2.5 \
                    cdot d\\'))
366             spacing_req.append(NoEscape(r'&= 2.5 \times ' + str(
                    bolt_dia_prov) + r'\\'))
367             spacing_req.append(NoEscape(r'&= ' + str(p_min) + r' \
                    text{ mm}\\'))
368             spacing_req.append(NoEscape(r'g_{\text{min}} &= 2.5 \
                    cdot d\\'))
369             spacing_req.append(NoEscape(r'&= 2.5 \times ' + str(
                    bolt_dia_prov) + r'\\'))
370             spacing_req.append(NoEscape(r'&= ' + str(g_min) + r' \
                    text{ mm}\\ \\'))
371             spacing_req.append(NoEscape(r'&[\text{Ref. Cl. 10.2.2}]
                    '))
372             spacing_req.append(NoEscape(r'\end{aligned}'))
373
374             spacing_prov = Math(inline=True)
375             spacing_prov.append(NoEscape(r'\begin{aligned}'))
376             spacing_prov.append(NoEscape(r'p_{\text{prov}} &= ' +
                    str(pitch) + r' \text{ mm}\\'))
377             spacing_prov.append(NoEscape(r'g_{\text{prov}} &= ' +
                    str(gauge) + r' \text{ mm}'))
378             spacing_prov.append(NoEscape(r'\end{aligned}'))
379
380             spacing_status = "PASS" if (pitch >= p_min and gauge >=
                     g_min) else "FAIL"
381             self.report_check.append(["Minimum Spacing",
                    spacing_req, spacing_prov, spacing_status])
382
383             # 2.3.2 Maximum Spacing (Cl. 10.2.3.1)
384             plate_thk_min = min(plate1_thk_raw, plate2_thk_raw)
385             p_max = as_int(min(32 * plate_thk_min, 300), 0)
386             g_max = as_int(min(32 * plate_thk_min, 300), 0)
387
388             max_spacing_req = Math(inline=True)
389             max_spacing_req.append(NoEscape(r'\begin{aligned}'))
390             max_spacing_req.append(NoEscape(r'p_{\text{max}} &= \
                    min(32 \cdot t, 300 \text{ mm})\\'))
391             max_spacing_req.append(NoEscape(r'&= \min(32 \times ' +
```

```
                str(plate_thk_min) + r', 300)\\'))
392             max_spacing_req.append(NoEscape(r'&= ' + str(p_max) + r
                ' \text{ mm}\\'))
393             max_spacing_req.append(NoEscape(r'g_{\text{max}} &= \
                min(32 \cdot t, 300 \text{ mm})\\'))
394             max_spacing_req.append(NoEscape(r'&= \min(32 \times ' +
                str(plate_thk_min) + r', 300)\\'))
395             max_spacing_req.append(NoEscape(r'&= ' + str(g_max) + r
                ' \text{ mm}\\ \\'))
396             max_spacing_req.append(NoEscape(r'&[\text{Ref. Cl.
                10.2.3.1}]'))
397             max_spacing_req.append(NoEscape(r'\end{aligned}'))
398
399             max_spacing_prov = Math(inline=True)
400             max_spacing_prov.append(NoEscape(r'\begin{aligned}'))
401             max_spacing_prov.append(NoEscape(r'p_{\text{prov}} &= '
                + str(pitch) + r' \text{ mm}\\'))
402             max_spacing_prov.append(NoEscape(r'g_{\text{prov}} &= '
                + str(gauge) + r' \text{ mm}'))
403             max_spacing_prov.append(NoEscape(r'\end{aligned}'))
404
405             max_spacing_status = "PASS" if (pitch <= p_max and
                gauge <= g_max) else "FAIL"
406             self.report_check.append(["Maximum Spacing",
                max_spacing_req, max_spacing_prov,
                max_spacing_status])
407
408             # 2.3.3 Edge Distance (Cl. 10.2.4)
409             bolt_hole_type_str = str(self.bolt.bolt_hole_type) if
                hasattr(self.bolt, 'bolt_hole_type') else "Standard"
410             d_hole = IS800_2007.cl_10_2_1_bolt_hole_size(d,
                bolt_hole_type_str)
411
412             if "Sheared" in edge_type or "hand flame cut" in
                edge_type:
413                 e_min_calc = f2(1.7 * d_hole, 0.0)
414                 e_min_multiplier = 1.7
415             else:  # Rolled, machine-flame cut, sawn and planed
416                 e_min_calc = f2(1.5 * d_hole, 0.0)
```

```
                e_min_multiplier = 1.5

            epsilon = math.sqrt(250 / fy)
            e_max_calc = f2(12 * plate_thk_min * epsilon, 0.0)

            edge_req = Math(inline=True)
            edge_req.append(NoEscape(r'\begin{aligned}'))
            edge_req.append(NoEscape(r'e_{\min} &= ' + str(
                e_min_multiplier) + r' \cdot d_0\\'))
            edge_req.append(NoEscape(r'&= ' + str(e_min_multiplier)
                + r' \times ' + f'{d_hole:.1f}' + r' = ' + f'{
                e_min_calc:.1f}' + r' \text{ mm}\\'))
            edge_req.append(NoEscape(r'e_{\text{max}} &= 12 \cdot t
                \cdot \varepsilon\\'))
            edge_req.append(NoEscape(r'&= 12 \times ' + str(
                plate_thk_min) + r' \times ' + f'{epsilon:.2f}' + r'
                \\'))
            edge_req.append(NoEscape(r'&= ' + str(e_max_calc) + r'
                \text{ mm}\\ \\'))
            edge_req.append(NoEscape(r'&[\text{Ref. Cl. 10.2.4}]'))
            edge_req.append(NoEscape(r'\end{aligned}'))

            edge_prov = Math(inline=True)
            edge_prov.append(NoEscape(r'e_{\text{prov}} = ' + str(
                e_dist) + r' \text{ mm}'))

            edge_status = "PASS" if (e_dist >= e_min_calc and
                e_dist <= e_max_calc) else "FAIL"
            self.report_check.append(["Edge Distance", edge_req,
                edge_prov, edge_status])

            #================================
            # Section 2.4: Number of Bolts
            #================================
            self.report_check.append([
                "SubSection", "Number of Bolts Required", "|p{4cm}|
                    p{5cm}|p{5.5cm}|p{1.5cm}|"
            ])

```

```
            bolts_req_initial = math.ceil(axial_kN / bolt_final_cap
                ) if bolt_final_cap > 0 else 0

            bolts_eq = Math(inline=True)
            bolts_eq.append(NoEscape(r'\begin{aligned}\\'))
            bolts_eq.append(NoEscape(r'n &= \frac{P}{V_{db}}\\\\'))
            bolts_eq.append(NoEscape(r'&= \frac{' + str(axial_kN) +
                r'}{' + str(bolt_final_cap) + r'}\\\\'))
            bolts_eq.append(NoEscape(r'&= ' + str(bolts_req_initial
                ) + r' \text{ nos.}\\'))
            bolts_eq.append(NoEscape(r'\end{aligned}'))

            self.report_check.append(["Bolts Required", f" {
                axial_kN:.2f} kN", bolts_eq, ""])

            #================================
            # Section 2.5: Bolt Arrangement
            #================================
            self.report_check.append([
                "SubSection", "Bolt Arrangement", "|p{4cm}|p{5cm}|p
                    {5.5cm}|p{1.5cm}|"
            ])

            self.report_check.append([
                "Bolt Pattern", "2", f"Arrangement: {rows} rows
                    {cols} columns", ""
            ])

            #================================
            # Section 2.6: Base Metal Strength
            #================================
            self.report_check.append([
                "SubSection", "Base Metal Strength", "|p{4cm}|p{5cm
                    }|p{5.5cm}|p{1.5cm}|"
            ])

            if is_comp:
                base_req = Math(inline=True)
                base_req.append(NoEscape(r'\begin{aligned}\\'))
```

```
477                 base_req.append(NoEscape(r'P_d &= \frac{A_g \cdot
                        f_y}{\gamma_{m0}}\\\\'))
478                 base_req.append(NoEscape(r'&= \frac{' + str(A_g) +
                        r' \times ' + str(fy) + r'}{1.10}\\\\'))
479                 base_req.append(NoEscape(r'&= ' + f'{
                        base_metal_capacity_kN:.2f}' + r' \text{ kN}\\')
                        )
480                 base_req.append(NoEscape(r'&[\text{Ref. Cl. 7.1.2}]
                        '))
481                 base_req.append(NoEscape(r'\end{aligned}'))
482
483                 base_status = "PASS" if base_metal_capacity_kN >=
                        axial_kN else "FAIL"
484                 self.report_check.append(["Plate Tension Capacity",
                         "", base_req, base_status])
485             else:
486                 # 1. Gross Section Yielding
487                 yield_req = Math(inline=True)
488                 yield_req.append(NoEscape(r'\begin{aligned}\\'))
489                 yield_req.append(NoEscape(r'T_{dg} &= \frac{A_g \
                        cdot f_y}{\gamma_{m0}}\\\\'))
490                 yield_req.append(NoEscape(r'&= \frac{' + str(A_g) +
                         r' \times ' + str(fy) + r'}{1.10}\\\\'))
491                 yield_req.append(NoEscape(r'&= ' + f'{T_dg:.2f}' +
                        r' \text{ kN}\\'))
492                 yield_req.append(NoEscape(r'&[\text{Ref. Cl. 6.2}]'
                        ))
493                 yield_req.append(NoEscape(r'\end{aligned}'))
494                 self.report_check.append(["Gross Section Yield", ""
                        , yield_req, ""])
495
496                 # 2. Net Section Rupture
497                 rup_req = Math(inline=True)
498                 rup_req.append(NoEscape(r'\begin{aligned}'))
499                 rup_req.append(NoEscape(r'T_{dn} &= \frac{0.9 A_n
                        f_u}{\gamma_{m1}}\\'))
500                 rup_req.append(NoEscape(r'&= ' + f'{T_dn:.2f}' + r'
                         \text{ kN}\\'))
501                 rup_req.append(NoEscape(r'&[\text{Ref. Cl. 6.3}]'))
```

```
                rup_req.append(NoEscape(r'\end{aligned}'))
                self.report_check.append(["Net Section Rupture", ""
                    , rup_req, ""])

                # 3. Block Shear (Cl 6.4)
                # Recalculate areas for report clarity
                # Note: We use the same logic as the solver's
                    check_base_metal_strength
                n_r = self.rows
                p = self.final_pitch
                g_val = self.final_gauge
                e_val = self.final_end_dist
                dia_hole = IS800_2007.cl_10_2_1_bolt_hole_size(d,
                    str(self.bolt.bolt_hole_type))

                t_min = min(plate1_thk_raw, plate2_thk_raw)

                Avg = t_min * ((n_r - 1) * g_val + e_val)
                Avn = t_min * ((n_r - 1) * g_val + e_val - (n_r -
                    0.5) * dia_hole)
                Atg = t_min * e_val
                Atn = t_min * (e_val - 0.5 * dia_hole)

                Tdb1 = (Avg * fy / (math.sqrt(3) * 1.10) + 0.9 *
                    Atn * fu / 1.25) / 1000
                Tdb2 = (0.9 * Avn * fu / (math.sqrt(3) * 1.25) +
                    Atg * fy / 1.10) / 1000
                Tdb = min(Tdb1, Tdb2)

                block_req = Math(inline=True)
                block_req.append(NoEscape(r'\begin{aligned}'))
                block_req.append(NoEscape(r'A_{vg} &= ' + f'{Avg:.0
                    f}' + r' \text{ mm}^2, \quad A_{vn} = ' + f'{Avn
                    :.0f}' + r' \text{ mm}^2\\'))
                block_req.append(NoEscape(r'A_{tg} &= ' + f'{Atg:.0
                    f}' + r' \text{ mm}^2, \quad A_{tn} = ' + f'{Atn
                    :.0f}' + r' \text{ mm}^2\\'))
                block_req.append(NoEscape(r'T_{db1} &= \frac{A_{vg}
                     f_y}{\sqrt{3} \gamma_{m0}} + \frac{0.9 A_{tn}
```

```
                    f_u}{\gamma_{m1}} = ' + f'{Tdb1:.2f}' + r' \text{ kN}\\'))
530                 block_req.append(NoEscape(r'T_{db2} &= \frac{0.9 A_{vn} f_u}{\sqrt{3} \gamma_{m1}} + \frac{A_{tg} f_y}{\gamma_{m0}} = ' + f'{Tdb2:.2f}' + r' \text{ kN}\\'))
531                 block_req.append(NoEscape(r'T_{db} &= \min(T_{db1}, T_{db2}) = ' + f'{Tdb:.2f}' + r' \text{ kN}\\'))
532                 block_req.append(NoEscape(r'&[\text{Ref. Cl. 6.4.1}]'))
533                 block_req.append(NoEscape(r'\end{aligned}'))
534                 self.report_check.append(["Block Shear", "", block_req, ""])
535
536                 # Governing Strength
537                 base_req = Math(inline=True)
538                 base_req.append(NoEscape(r'\begin{aligned}'))
539                 base_req.append(NoEscape(r'T_d &= \min(T_{dg}, T_{dn}, T_{db})\\'))
540                 base_req.append(NoEscape(r'&= ' + f'{base_metal_capacity_kN:.2f}' + r' \text{ kN}\\'))
541                 base_req.append(NoEscape(r'\end{aligned}'))
542
543                 base_status = "PASS" if base_metal_capacity_kN >= axial_kN else "FAIL"
544                 self.report_check.append(["Plate Tension Capacity", "", base_req, base_status])
545
546             #============================
547             # Section 2.7: Design Summary
548             #============================
549             self.report_check.append([
550                 "SubSection", "Design Summary", "|p{4cm}|p{5cm}|p{5.5cm}|p{1.5cm}|"
551             ])
552
553             bolt_capacity_total = f2(bolt_final_cap * n_bolts, 0.0)
```

```
554             bolt_ur = axial_kN / bolt_capacity_total if
                    bolt_capacity_total > 0 else 999.0
555
556             plate_ur = axial_kN / base_metal_capacity_kN if
                    base_metal_capacity_kN > 0 else 999.0
557
558             # Overall UR is max of both
559             overall_ur_val = max(bolt_ur, plate_ur)
560             overall_ur = round(overall_ur_val, 3)
561
562             ur_req = Math(inline=True)
563             ur_req.append(NoEscape(r'\begin{aligned}\\'))
564             ur_req.append(NoEscape(r'\text{Bolt Capacity} &= ' +
                    str(bolt_capacity_total) + r' \text{ kN}\\'))
565             ur_req.append(NoEscape(r'\text{Plate Capacity} &= ' +
                    str(base_metal_capacity_kN) + r' \text{ kN}\\\\'))
566             ur_req.append(NoEscape(r'\text{UR}_{\text{bolt}} &= \
                    frac{' + str(axial_kN) + r'}{' + str(
                    bolt_capacity_total) + r'}\\'))
567             ur_req.append(NoEscape(r'&= ' + f'{bolt_ur:.3f}' + r'
                    \\\\'))
568             ur_req.append(NoEscape(r'\text{UR}_{\text{plate}} &= \
                    frac{' + str(axial_kN) + r'}{' + str(
                    base_metal_capacity_kN) + r'}\\'))
569             ur_req.append(NoEscape(r'&= ' + f'{plate_ur:.3f}' + r'
                    \\\\'))
570             ur_req.append(NoEscape(r'\text{UR}_{\text{final}} &= \
                    max(\text{UR}_{\text{bolt}}, \text{UR}_{\text{plate
                    }})\\'))
571             ur_req.append(NoEscape(r'&= ' + str(overall_ur) + r'\
                    end{aligned}'))
572
573             util_prov = Math(inline=True)
574             util_prov.append(NoEscape(str(overall_ur) + r' \leq 1.0
                    '))
575
576             util_status = "PASS" if overall_ur_val <= 1.0 else "
                    FAIL"
577             self.report_check.append(["Utilization Ratio", f"{
```

```
                axial_kN:.2f} kN", ur_req, util_status])
578
579             Disp_2d_image = []
580             Disp_3D_image = "/ResourceFiles/images/3d.png"
581             rel_path = os.path.abspath(".").replace("\\", "/")
582             fname_no_ext = popup_summary.get("filename", "
                    LapJointBoltedReport")
583             folder = popup_summary.get('folder', './reports')
584             os.makedirs(folder, exist_ok=True)
585
586             CreateLatex.save_latex(
587                 CreateLatex(), self.report_input, self.report_check
                        ,
588                 popup_summary, fname_no_ext, rel_path,
                        Disp_2d_image, Disp_3D_image,
589                 module=self.module
590             )
591
592             self.logger.info(f"Report generated successfully: {
                    fname_no_ext}.pdf")
593             return True
594
595         except Exception as e:
596             print(f"WARNING in save_design(): {e}")
597             import traceback
598             traceback.print_exc()
599             return False
600 %-------------------- end code ---------------
```

3.3.1 Explanation of the Code

The `save_design` function in the Bolted Lap Joint module serves as the core automation engine for generating the final design report. It orchestrates the entire process from data retrieval to PDF generation. The function's key operations are summarized below:

1. **Data Retrieval and Initialization:** The function first extracts the design

parameters stored in the class instance. This includes the tensile or compressive load (P), plate dimensions (t_1, t_2, w), and bolt properties (diameter d, grade, and hole type). It also identifies the connection mode (Bearing or Friction Grip) to apply the correct design standards.

2. **Design Status Verification:** A critical check is performed on the `design_status` attribute.
   - If the design is marked as **False** (failed), the function generates a brief report summarizing the failure (e.g., "Bolt capacity zero" or "Plate utilization ¿ 1.0").
   - If **True** (safe), it proceeds to compile the full report with all calculation details.
3. **Input Dictionary Construction:** The function builds the `report_input` dictionary, which acts as the data bridge to the LaTeX template. Key entries include:
   - **Geometry:** Thickness of connected plates and overlap dimensions.
   - **Bolt Data:** Shear capacity (V_{dsb}) and bearing capacity (V_{dpb}) per bolt.
   - **Arrangement:** Number of rows, columns, pitch (p), and gauge (g).
4. **Automated LaTeX Code Generation:** Using the `pylatex` library, the function dynamically creates LaTeX equations. It injects Python variables directly into math strings to show the step-by-step verification:
   - **Shear Check:** $V_{dsb} = \frac{f_u \times A_s}{\sqrt{3} \times \gamma_{mb}}$
   - **Bearing Check:** $V_{dpb} = 2.5 \times k_b \times d \times t \times f_u$
   - **Utilization Ratio:** $UR = \frac{F_{applied}}{F_{capacity}}$

   This ensures the report reflects the exact state of the current design iteration.
5. **Report Compilation and Export:** Finally, the function calls `CreateLatex.save_latex` to render the complete document. It handles file paths robustly using `os.path` to ensure cross-platform compatibility and automatically embeds 2D/3D images of the joint configuration.

3.4 Documentation

The directory structure used for the Report Generation is organized as follows:

```
src/osdag_core

 design_report/
    reportGenerator_latex.py

 design_type/
    connection/
       lap_joint_bolted.py

Report_functions.py
```

Program Start Calls To start the program, open the Osdag folder, go to src and start the terminal with that path, execute the following command from the project root:

```
$ python -m osdag_gui
```

3.5 References

– Osdag Github repository: https://osdag.github.io/

– IS 800:2007, "General Construction in Steel Code of Practice of Practice"

Chapter 4

Report Generation of Bolted Butt Joint

4.1 Problem Statement

The primary objective of this task was to generate a comprehensive design report for **Bolted Butt Joint Connections**. Osdag (Open Steel Design and Graphics) is an open-source tool designed for the design and detailing of steel structures in accordance with Indian Standards. A critical requirement for steel structures is ensuring they pass all Design and Detailing Checklists (DDCL) to guarantee durability and safety. The challenge involved meticulously converting all necessary steps, formulas, and compliance checks stipulated by IS 800:2007 into code to ensure successful report generation for butt joint connections, which involve complex interactions between main plates, cover plates, and bolts.

4.2 Tasks Done

4.2.1 Core Save Design Function Development

* **Implemented comprehensive `save_design()` function in `butt_joint_bolted.py` with robust error handling**

  Developed the main orchestrating function that serves as the entry point

for report generation. This function coordinates data collection from the UI, validates all design parameters (including cover plate thickness and packing plates), performs calculations, and triggers the LaTeX report compilation. It includes comprehensive error handling to manage exceptions during file operations and data processing.

* **Integrated with existing Osdag LaTeX report infrastructure using `CreateLatex` class**

  Successfully interfaced the new bolted butt joint module with Osdag's established LaTeX report generation framework. This required constructing a detailed `report_input` dictionary and a structured `report_check` list to pass data dynamically to the `CreateLatex.save_latex` method, ensuring consistency with other connection modules.

* **Implemented proper file path handling with OS-independent path management**

  Coded cross-platform file path management using Python's `os.path` module to ensure the report generation works seamlessly across different operating systems. The function automatically handles directory creation for reports if they do not exist.

* **Report Input Dictionary Structure**

  Architected a complete data structure that captures every piece of information needed for report generation, including:

  - Module identification (Butt Joint Bolted Connection)
  - Material properties (Ultimate and Yield stress)
  - Geometric parameters (Main plate thickness, Cover plate thickness)
  - Bolt specifics (Diameter, Grade, Type - Friction/Bearing)
  - Load information (Tensile/Compression Force)
  - Cover Plate details (Single/Double, Thickness)

4.2.2 Detailed Design Verification and Check Implementation

Cover Plate and Packing Plate Checks

* **Implemented Cover Plate Thickness Logic**

  Coded logic to automatically determine required cover plate thickness based on the main plate thickness (T_{min}) and connection type (Single/-Double Cover) [Cl. 10.3.3.3]:

$$T_{cp} \geq \begin{cases} \frac{5}{8}T_{min} & \text{for Single Cover} \\ \frac{9}{16}T_{min} & \text{for Double Cover} \end{cases} \tag{4.1}$$

* **Packing Plate and Reduction Factor (β_{pkg})**

  Implemented checks for packing plates when connecting plates of different thicknesses. If the packing thickness exceeds 6mm, a reduction factor (β_{pkg}) is applied to the bolt shear capacity [Cl. 10.3.3.3]:

$$\beta_{pkg} = 1 - 0.0125 \times t_{pkg} \tag{4.2}$$

Bolt Capacity Checks

* **Implemented Bolt Shear and Bearing calculations**

  Coded the mathematical logic for determining bolt capacity based on IS 800:2007 requirements [Cl. 10.3.3, 10.3.4]. The function dynamically calculates capacity based on the bolt type (Bearing or Friction Grip).

  Shear Capacity:

$$V_{dsb} = \frac{f_{ub} \times A_{sb}}{\gamma_{mb}} \tag{4.3}$$

  Bearing Capacity:

$$V_{dpb} = \frac{2.5 \times k_b \times d \times t \times f_u}{\gamma_{mb}} \tag{4.4}$$

* **Capacity Reduction Factors**

  Implemented advanced checks for "Long Joints" (β_{lj}) and "Large Grips"

(β_{lg}) as per Cl. 10.3.3.1 and 10.3.3.2. The code automatically reduces bolt capacity if the joint length or grip length exceeds specified limits:

$$\beta_{lj} = 1.075 - \frac{l_j}{200d} \quad (0.75 \leq \beta_{lj} \leq 1.0) \tag{4.5}$$

Base Metal Strength Analysis

* **Implemented dual strength checks (Yielding and Rupture)**

  Developed parallel calculation paths for both failure modes in tension member design [Cl. 6.2, 6.3]:

  Gross Yielding:

$$T_{dg} = \frac{A_g \times f_y}{\gamma_{m0}} \tag{4.6}$$

  Net Rupture:

$$T_{dn} = \frac{0.9 \times A_n \times f_u}{\gamma_{m1}} \tag{4.7}$$

* **Block Shear Check**

  Integrated Block Shear Failure checks (V_{db}) [Cl. 6.4] to ensure the connection does not fail by tearing out a block of material. The final design strength (T_{db}) is taken as the minimum of yielding, rupture, and block shear capacities.

4.2.3 Advanced LaTeX Mathematical Formatting

Mathematical Equation Integration

* **Used `NoEscape` LaTeX formatting for complex mathematical expressions**

  Mastered the `pylatex.utils.NoEscape` functionality to render complex mathematical equations without Python string escaping issues. This was crucial for rendering fractions, subscripts, and Greek letters (e.g., γ_{mb}, π) correctly in the final PDF.

* **Dynamic Equation Generation**

  Implemented formatted strings (f-strings) within the LaTeX code to inject calculated Python variables (like `bolt_shear_kN`, `kb`, `An`) directly into

the equation display strings. This ensures the report shows the actual numerical substitution steps, not just the final result.

4.2.4 Design Calculations Integration

Utilization Ratio and Detailing

* **Calculated Utilization Ratio (UR)**

  Implemented a tracking system to calculate the utilization ratio for both the bolts and the base metal. The code identifies the critical UR to determine the overall safety of the connection:

$$UR = \frac{\text{Applied Force}}{\text{Design Capacity}} \tag{4.8}$$

* **Detailing Requirements Implementation**

  Implemented practical construction detailing checks as per IS 800:2007 [Cl. 10.2], verifying:

  - Minimum Pitch and Gauge ($2.5 \times d$)
  - Minimum and Maximum Edge Distances

  The `save_design` function automatically verifies if provided spacing (p_{prov}, e_{prov}) meets the calculated limits (p_{min}, e_{min}) and reports a "PASS" or "FAIL" status in the final document.

4.3 Python Code

The entire logic for the report generation of Bolted Butt Joint module is within the save_design() function. And it's being connected to the main GUI through ui_template.py and ui_popup_summary.py files. Below are key snippets from the module with explanations of their functionality.

Description of the Script

The script defines the save_design() function, which handles everything from user input to final design report generation. It uses reportGenerator_latex.py

to manage properties and perform calculations.

Python Code

The following snippets showcase the core logic implemented in the report generation task.

Listing 4.1: Bolted Butt Joint save_design() function

```
%--------------------begin code-------------
def save_design(self, popup_summary):
        """
        Generate the LaTeX design report for Lap Joint Bolted
           Connection (Tension/Compression)
        per IS 800:2007.
        """
        try:
            def g(attr, default=None):
                v = getattr(self, attr, default)
                return default if v is None else v

            def f2(x, default=0.0):
                try:
                    return round(float(x), 2)
                except (TypeError, ValueError):
                    return default

            def as_int(x, default=0):
                try:
                    return int(round(float(x)))
                except (TypeError, ValueError):
                    return default

            if not getattr(self, 'design_status', False):
                self.report_input = {
                    KEY_MODULE: "Butt Joint Bolted Connection",
                    KEY_MAIN_MODULE: "Plate(d) Connection",
                    "Design Status": "TITLE",
                    "Status": "Design not completed
                       successfully.",
```

```
30                  }
31                  self.report_check = []
32                  self.report_check.append([
33                      "SubSection", "Design Status", "|p{4cm}|p{5
                            cm}|p{5.5cm}|p{1.5cm}|"
34                  ])
35                  self.report_check.append(["Design", "Design not
                        completed successfully.", "", "FAIL"])
36
37                  Disp_2d_image = []
38                  Disp_3D_image = "/ResourceFiles/images/3d.png"
39                  rel_path = os.path.abspath(".").replace("\\", "
                        /")
40                  fname_no_ext = popup_summary.get("filename", "
                        ButtJointBoltedReport")
41                  folder = popup_summary.get('folder', './reports
                        ')
42                  os.makedirs(folder, exist_ok=True)
43
44                  CreateLatex.save_latex(
45                      CreateLatex(), self.report_input, self.
                            report_check,
46                      popup_summary, fname_no_ext, rel_path,
                            Disp_2d_image, Disp_3D_image,
47                      module=getattr(self, 'module', 'Butt Joint
                            Bolted')
48                  )
49                  return True
50
51              self.module = g('module', 'Butt Joint Bolted')
52              self.mainmodule = 'Plate(d) Connection'
53              design_for = str(g('design_for', 'Tension')).strip
                    ()
54              is_comp = design_for.lower().startswith('c')
55
56              plate1_thk = f2(g('plate1thk', g('pltthk', 0.0)),
                    0.0)
57              plate2_thk = f2(g('plate2thk', g('pltthk', 0.0)),
                    0.0)
```

```
58              width = f2(g('width', 0.0), 0.0)
59              axial_kN = f2(g('axial_force_kN', g('tensile_force'
                    , 0.0)), 0.0)
60
61              edge_type = getattr(self.bolt, 'edgetype', 'Sheared
                     or hand flame cut')
62              bolt_dia_prov = f2(getattr(self.bolt, '
                    bolt_diameter_provided', 0.0) if hasattr(self,
                    'bolt') else 0.0, 0.0)
63              bolt_grade_prov = f2(getattr(self.bolt, '
                    bolt_grade_provided', 0.0) if hasattr(self, '
                    bolt') else 0.0, 0.0)
64              bolt_type = getattr(self.bolt, 'bolt_type',
                    VALUE_NOT_APPLICABLE) if hasattr(self, 'bolt')
                    else VALUE_NOT_APPLICABLE
65
66              bolt_shear_kN = f2(getattr(self.bolt, '
                    bolt_shear_capacity', 0.0) if hasattr(self, '
                    bolt') else 0.0, 0.0)
67              bolt_bearing_kN = f2(getattr(self.bolt, '
                    bolt_bearing_capacity', 0.0) if hasattr(self, '
                    bolt') else 0.0, 0.0)
68              bolt_final_cap = f2(getattr(self.bolt, '
                    bolt_capacity', 0.0) if hasattr(self, 'bolt')
                    else 0.0, 0.0)
69
70              rows = as_int(g('rows', 0), 0)
71              cols = as_int(g('cols', 0), 0)
72              n_bolts = as_int(g('number_bolts', 0), 0)
73              pitch = f2(g('final_pitch', 0.0), 0.0)
74              gauge = f2(g('final_gauge', 0.0), 0.0)
75              e_dist = f2(g('final_edge_dist', 0.0), 0.0)
76
77              t_fu_fy_list = getattr(self, '
                    bolt_conn_plates_t_fu_fy', [])
78              if t_fu_fy_list and len(t_fu_fy_list) > 0:
79                  fu = t_fu_fy_list[0][1] if len(t_fu_fy_list[0])
                         > 1 else 0
80                  fy = t_fu_fy_list[0][2] if len(t_fu_fy_list[0])
```

```
                    > 2 else 0
            else:
                fy = g('yield_stress', 0)
                fu = 0

            base_metal_capacity_kN = f2(g('
                base_metal_capacity_kN', 0.0), 0.0)

            A_g = f2(g('A_g', 0.0), 0.0)
            T_dg = f2(g('T_dg', 0.0), 0.0)
            T_dn = f2(g('T_dn', 0.0), 0.0)
            T_db = f2(g('T_db', 0.0), 0.0)

            overall_ur = round(g('utilization_ratio', 0.0), 3)

            self.report_input = {
                KEY_MODULE: "Butt Joint Bolted Connection",
                KEY_MAIN_MODULE: "Plate(d) Connection",
                KEY_DISP_DESIGN_FOR: design_for,
                "Thickness of Plate-1 (mm) *": plate1_thk,
                "Thickness of Plate-2 (mm) *": plate2_thk,
                "Width of Plate (mm) *": width,
                "Material *": getattr(self, 'main_material',
                    VALUE_NOT_APPLICABLE),
                "Diameter (mm) *": bolt_dia_prov,
                "Property Class *": bolt_grade_prov,
                "Type *": bolt_type,
                f"{'Tensile' if not is_comp else 'Axial'} Force
                    (kN) *": axial_kN,
                "Additional inputs": "TITLE",
                "Bolt Hole Type": getattr(self.bolt, '
                    boltholetype', 'Standard'),
                "Slip Factor ( f )": getattr(self.bolt, 'mu_f',
                    'N/A'),
                "Edge Preparation Method": edge_type
            }

            self.report_check = []

```

```
114             #
                    ============================================================

115             #=========== SECTION 1: DESIGN OF COVER PLATES
                    ===============
116             #
                    ============================================================

117             self.report_check.append([
118                 "SubSection", "Design of Cover Plates", "|p{4cm
                        }|p{5cm}|p{5.5cm}|p{1.5cm}|"
119             ])
120
121             # 1.1 Cover Plate Thickness (Cl. 10.5.2.3 and
                    10.5.2.4 logic adapted)
122             t_min = min(plate1_thk, plate2_thk)
123             cover_plate_type = str(g('cover_plate_type', '
                    Double Cover Plate'))
124
125             cp_req = Math(inline=True)
126             cp_req.append(NoEscape(r'\begin{aligned}'))
127
128             if "double" in cover_plate_type.lower():
129                 t_req = math.ceil(1.125 * t_min) # 9/8 * t_min
130                 cp_req.append(NoEscape(r'\text{For Double Cover
                        Plates:}\\'))
131                 cp_req.append(NoEscape(r't_{cp, req} &= \frac
                        {9}{8} \cdot t_{\min}\\'))
132                 cp_req.append(NoEscape(r'&= \frac{9}{8} \times
                        ' + str(t_min) + r'\\'))
133                 cp_req.append(NoEscape(r'&= ' + str(t_req) + r'
                        \text{ mm}\\'))
134             else: # Single Cover Plate
135                 t_req = math.ceil(0.625 * t_min) # 5/8 * t_min
136                 cp_req.append(NoEscape(r'\text{For Single Cover
                        Plate:}\\'))
137                 cp_req.append(NoEscape(r't_{cp, req} &= \frac
                        {5}{8} \cdot t_{\min}\\'))
138                 cp_req.append(NoEscape(r'&= \frac{5}{8} \times
```

```
                    ' + str(t_min) + r'\\'))
139                 cp_req.append(NoEscape(r'&= ' + str(t_req) + r'
                        \text{ mm}\\'))
140
141             cp_req.append(NoEscape(r'\end{aligned}'))
142
143             t_cp_prov = float(self.
                    calculated_cover_plate_thickness) if hasattr(
                    self, 'calculated_cover_plate_thickness') else
                    t_req
144             cp_prov = Math(inline=True)
145             cp_prov.append(NoEscape(r't_{cp, prov} = ' + str(
                    t_cp_prov) + r' \text{ mm}'))
146
147             cp_status = "PASS" if t_cp_prov >= t_req else "FAIL
                    "
148             self.report_check.append(["Cover Plate Thickness",
                    cp_req, cp_prov, cp_status])
149
150             # 1.2 Packing Plate (Cl. 10.3.3.2)
151             packing_thk = float(getattr(self, '
                    packing_plate_thickness', 0.0))
152             if packing_thk > 0:
153                 pack_req = Math(inline=True)
154                 pack_req.append(NoEscape(r'\begin{aligned}'))
155                 pack_req.append(NoEscape(r'\text{Difference in
                        plate thickness:}\\'))
156                 pack_req.append(NoEscape(r't_{pkg} &= |t_1 -
                        t_2|\\'))
157                 pack_req.append(NoEscape(r'&= |' + str(
                        plate1_thk) + ' - ' + str(plate2_thk) + '
                        |\\'))
158                 pack_req.append(NoEscape(r'&= ' + str(
                        packing_thk) + r' \text{ mm}\\'))
159                 pack_req.append(NoEscape(r'\end{aligned}'))
160
161                 pack_prov = Math(inline=True)
162                 pack_prov.append(NoEscape(r't_{pkg, prov} = ' +
                        str(packing_thk) + r' \text{ mm}'))
```

```
163
164                 self.report_check.append(["Packing Plate",
                        pack_req, pack_prov, "INFO"])
165
166             #
                    ============================================================
167             #=========== SECTION 2.1: CALCULATING BOLT STRENGTH
                    ==========
168             #
                    ============================================================
169             self.report_check.append([
170                 "SubSection", "Calculating Bolt Strength", "|p
                        {4cm}|p{5cm}|p{5.5cm}|p{1.5cm}|"
171             ])
172
173             d = float(self.bolt.bolt_diameter_provided)
174             bolt_grade = float(self.bolt.bolt_grade_provided)
175             f_ub = int(bolt_grade * 100)
176
177             plate1_thk_raw = float(self.plate1.thickness[0]) if
                    isinstance(self.plate1.thickness, list) else
                   float(self.plate1.thickness)
178             plate2_thk_raw = float(self.plate2.thickness[0]) if
                    isinstance(self.plate2.thickness, list) else
                   float(self.plate2.thickness)
179
180             bolt_shank_area = f2(math.pi * d**2 / 4, 0.0)
181
182             if hasattr(self.bolt, 'bolt_net_area_provided'):
183                 bolt_net_area = f2(self.bolt.
                        bolt_net_area_provided, 0.0)
184             else:
185                 bolt_net_area = f2(math.pi * (d - 0.9382 * math
                        .sqrt(d))**2 / 4, 0.0)
186
187             gamma_mb = 1.25
188
```

```
189             if self.bolt.bolt_type != "Bearing Bolt":
190                 # ========== FRICTION GRIP TYPE BOLTING (Cl.
                        10.4.3) ==========
191                 f_0 = 0.7 * f_ub
192                 F_o = bolt_net_area * f_0
193                 mu = float(self.bolt.mu_f) if hasattr(self.bolt
                        , 'mu_f') else 0.3
194                 n_e = 1  # Number of effective interfaces (
                        single lap joint)
195 
196                 bolt_hole_type_str = str(self.bolt.
                        bolt_hole_type) if hasattr(self.bolt, '
                        bolt_hole_type') else "Standard"
197                 d_0 = IS800_2007.cl_10_2_1_bolt_hole_size(d,
                        bolt_hole_type_str)
198 
199                 hole_type_lower = bolt_hole_type_str.lower()
200                 if "standard" in hole_type_lower:
201                     K_h = 1.0
202                 elif "over" in hole_type_lower or "short" in
                        hole_type_lower:
203                     K_h = 0.85
204                 else:  # long slotted
205                     K_h = 0.7
206 
207                 V_nsf = mu * n_e * K_h * F_o
208                 gamma_mf = 1.25  # For ultimate load
209                 V_dsf_theoretical = V_nsf / gamma_mf
210                 V_dsf_kN_theoretical = V_dsf_theoretical / 1000
211 
212                 slip_req = Math(inline=True)
213                 slip_req.append(NoEscape(r'\begin{aligned}'))
214                 slip_req.append(NoEscape(r'V_{dsf} &= \frac{V_{
                        nsf}}{\gamma_{mf}}\\'))
215                 slip_req.append(NoEscape(r'V_{nsf} &= \mu \cdot
                         n_e \cdot K_h \cdot F_o\\'))
216                 slip_req.append(NoEscape(r'\mu &= ' + f'{mu:.2f
                        }' + r' \text{ (slip factor)}\\'))
217                 slip_req.append(NoEscape(r'n_e &= ' + str(n_e)
```

```
                    + r' \text{ (interfaces)}\\'))
218                 slip_req.append(NoEscape(r'K_h &= ' + f'{K_h:.2
                    f}' + r' \text{ (hole factor)}\\'))
219                 slip_req.append(NoEscape(r'f_0 &= 0.7 f_{ub} =
                    ' + f'{f_0:.1f}' + r' \text{ MPa}\\'))
220                 slip_req.append(NoEscape(r'A_{nb} &= ' + f'{
                    bolt_net_area:.2f}' + r' \text{ mm}^2\\'))
221                 slip_req.append(NoEscape(r'F_o &= A_{nb} \times
                     f_0 = ' + f'{F_o:.2f}' + r' \text{ N}\\'))
222                 slip_req.append(NoEscape(r'V_{nsf} &= ' + f'{mu
                    :.2f}' + r' \times ' + str(n_e) + r' \times
                     ' + f'{K_h:.2f}' + r' \times ' + f'{F_o:.2
                    f}' + r'\\'))
223                 slip_req.append(NoEscape(r'&= ' + f'{V_nsf:.2f}
                    ' + r' \text{ N}\\'))
224                 slip_req.append(NoEscape(r'V_{dsf} &= \frac{' +
                     f'{V_nsf:.2f}' + r'}{' + str(gamma_mf) + r
                    '} = ' + f'{V_dsf_kN_theoretical:.2f}' + r'
                     \text{ kN}\\'))
225                 slip_req.append(NoEscape(r'&[\text{Ref. Cl.
                    10.4.3}]'))
226                 slip_req.append(NoEscape(r'\end{aligned}'))
227
228                 self.report_check.append(["Slip Resistance", ""
                    , slip_req, ""])
229
230             else:  # Bearing Bolt
231                 # ========== SHEAR CAPACITY (Cl. 10.3.3)
                    ==========
232                 # Strategy: Use the Solver's final Shear
                    Capacity (bolt_shear_kN) as the source of
                    truth to ensure Report matches Dock.
233                 # Back-calculate the Effective Area (A_eff)
                    that yields this capacity, then display it
                    in the formula.
234                 # This handles cases where Solver uses
                    different Area assumptions (e.g. shank vs
                    net) or different reduction factors.
235
```

```
                V_dsb_kN = bolt_shear_kN
                V_nsb_val = V_dsb_kN * gamma_mb

                try:
                    vals = str(bolt_grade_prov).split('.')
                    if len(vals) >= 2:
                        f_ub_val = int(vals[0]) * 100
                    else:
                        f_ub_val = 400
                except (ValueError, TypeError, IndexError):
                    f_ub_val = 400

                n_n = self.planes if hasattr(self, 'planes')
                    else 1

                # Back-calculate effective area per bolt per
                    plane (forcing n_s=0 for display simplicity
                    )
                # V_nsb = (f_ub / sqrt(3)) * (n_n * A_eff)
                if n_n > 0 and f_ub_val > 0:
                     A_eff = (V_nsb_val * 1000.0 * math.sqrt
                         (3.0)) / (f_ub_val * n_n)
                else:
                     A_eff = 0.0

                shear_req = Math(inline=True)
                shear_req.append(NoEscape(r'\begin{aligned}\\')
                    )
                shear_req.append(NoEscape(r'V_{dsb} &= \frac{V_
                    {nsb}}{\gamma_{mb}}\\\\'))
                shear_req.append(NoEscape(r'V_{nsb} &= \frac{f_
                    {ub}}{\sqrt{3}} \cdot (n_n \cdot A_{nb} +
                    n_s \cdot A_{sb})\\'))
                shear_req.append(NoEscape(r'&= \frac{' + str(
                    f_ub_val) + r'}{\sqrt{3}} \times (' + str(
                    n_n) + r' \times ' + f'{A_eff:.2f}' + r' +
                    0)\\'))
                shear_req.append(NoEscape(r'&= ' + f'{V_nsb_val
                    :.2f}' + r' \text{ kN}\\\\'))
```

```
263                 shear_req.append(NoEscape(r'V_{dsb} &= \frac{'
                        + f'{V_nsb_val:.2f}' + r'}{' + str(gamma_mb
                        ) + r'}\\'))
264                 shear_req.append(NoEscape(r'&= ' + f'{V_dsb_kN
                        :.2f}' + r' \text{ kN}\\'))
265                 shear_req.append(NoEscape(r'&[\text{Ref. Cl.
                        10.3.3}]'))
266                 shear_req.append(NoEscape(r'\end{aligned}'))
267
268                 self.report_check.append(["Shear Capacity", "",
                         shear_req, ""])
269
270                 # ========== BEARING CAPACITY (Cl. 10.3.4)
                        ==========
271                 V_dpb_kN = bolt_bearing_kN
272                 V_npb = V_dpb_kN * gamma_mb
273
274                 t_min = min(plate1_thk_raw, plate2_thk_raw)
275                 f_u_plate = min(self.plate1.fu, self.plate2.fu)
276
277                 bolt_hole_type_str = str(self.bolt.
                        bolt_hole_type) if hasattr(self.bolt, '
                        bolt_hole_type') else "Standard"
278                 d_0 = IS800_2007.cl_10_2_1_bolt_hole_size(d,
                        bolt_hole_type_str)
279
280                 e = float(self.final_end_dist) if hasattr(self,
                         'final_end_dist') and self.final_end_dist
                        > 0 else float(self.bolt.min_end_dist_round
                        )
281                 p = float(self.final_pitch) if hasattr(self, '
                        final_pitch') and self.final_pitch > 0 else
                         float(self.bolt.min_pitch_round)
282
283                 # Calculate kb factor components (always
                        calculate for report display)
284                 if p > 0:
285                     kb_1 = e / (3.0 * d_0)
286                     kb_2 = p / (3.0 * d_0) - 0.25
```

```
287                         kb_3 = f_ub / f_u_plate
288                         kb_4 = 1.0
289                         kb_calc = min(kb_1, kb_2, kb_3, kb_4)
290                     else:
291                         kb_1 = e / (3.0 * d_0)
292                         kb_2 = float('inf')  # Not applicable
293                         kb_3 = f_ub / f_u_plate
294                         kb_4 = 1.0
295                         kb_calc = min(kb_1, kb_3, kb_4)
296
297                     if hasattr(self.bolt, 'kb') and self.bolt.kb is
                            not None:
298                         k_b = f2(self.bolt.kb, 1.0)
299                     else:
300                         k_b = f2(kb_calc, 1.0)
301
302                     kb_req = Math(inline=True)
303                     kb_req.append(NoEscape(r'\begin{aligned}\\'))
304                     kb_req.append(NoEscape(r'k_b &= \min\left(\frac
                            {e}{3d_0}, \frac{p}{3d_0}-0.25, \frac{f_{ub
                            }}{f_u}, 1.0\right)\\\\'))
305                     kb_req.append(NoEscape(r'&= \min\left(\frac{' +
                            f'{e:.1f}' + r'}{3 \times ' + f'{d_0:.1f}'
                            + r'}, \frac{' + f'{p:.1f}' + r'}{3 \times
                            ' + f'{d_0:.1f}' + r'}-0.25, \frac{' + str
                            (f_ub) + r'}{' + str(f_u_plate) + r'}, 1.0\
                            right)\\\\'))
306
307                     if p > 0:
308                         kb_req.append(NoEscape(r'&= \min(' + f'{
                                kb_1:.2f}' + r', ' + f'{kb_2:.2f}' + r'
                                , ' + f'{kb_3:.2f}' + r', 1.0)\\\\'))
309                     else:
310                         kb_req.append(NoEscape(r'&= \min(' + f'{
                                kb_1:.2f}' + r', ' + f'{kb_3:.2f}' + r'
                                , 1.0)\\\\'))
311
312                     kb_req.append(NoEscape(r'&= ' + f'{k_b:.2f}'))
313                     kb_req.append(NoEscape(r'\end{aligned}'))
```

```
314                 self.report_check.append(["Kb Factor", "",
                        kb_req, ""])
315
316                 bearing_req = Math(inline=True)
317                 bearing_req.append(NoEscape(r'\begin{aligned}\\
                        '))
318                 bearing_req.append(NoEscape(r'V_{dpb} &= \frac{
                        V_{npb}}{\gamma_{mb}}\\\\'))
319                 bearing_req.append(NoEscape(r'V_{npb} &= 2.5 \
                        cdot k_b \cdot d \cdot t \cdot f_u\\'))
320                 bearing_req.append(NoEscape(r'&= 2.5 \times ' +
                         f'{k_b:.3f}' + r' \times ' + f'{d:.1f}' +
                        r' \times ' + f'{t_min:.1f}' + r' \times '
                        + str(f_u_plate) + r'\\'))
321                 bearing_req.append(NoEscape(r'&= ' + f'{V_npb
                        :.2f}' + r' \text{ kN}\\\\'))
322                 bearing_req.append(NoEscape(r'V_{dpb} &= \frac{
                        ' + f'{V_npb:.2f}' + r'}{' + f'{gamma_mb:.2
                        f}' + r'}\\'))
323                 bearing_req.append(NoEscape(r'&= ' + f'{
                        V_dpb_kN:.2f}' + r' \text{ kN}\\'))
324                 bearing_req.append(NoEscape(r'&[\text{Ref. Cl.
                        10.3.4}]'))
325                 bearing_req.append(NoEscape(r'\end{aligned}'))
326
327                 self.report_check.append(["Bearing Capacity", "
                        ", bearing_req, ""])
328
329                 V_db_kN = bolt_final_cap
330
331                 cap_req = Math(inline=True)
332                 cap_req.append(NoEscape(r'\begin{aligned}'))
333                 cap_req.append(NoEscape(r'V_{db} &= \min(' + f'
                        {bolt_shear_kN:.2f}' + r', ' + f'{
                        bolt_bearing_kN:.2f}' + r')'+r'\\'))
334                 cap_req.append(NoEscape(r'&= ' + f'{V_db_kN:.2f
                        }' + r' \text{ kN}'+r'\\'))
335                 cap_req.append(NoEscape(r'&[\text{Ref. Cl.
                        10.3.2, IS 800:2007}]'))
```

```
336                 cap_req.append(NoEscape(r'\end{aligned}'))
337
338                 self.report_check.append(["Bolt Design Capacity
                        ", "", cap_req, ""])
339
340             #
                    ============================================================
341             #=========== SECTION 2.2: REDUCTION FACTORS
                    ===========
342             #
                    ============================================================
343             self.report_check.append([
344                 "SubSection", "Reduction Factors", "|p{4cm}|p{5
                        cm}|p{5.5cm}|p{1.5cm}|"
345             ])
346
347             l_j = (self.rows - 1) * self.final_pitch if self.
                    rows > 1 else 0
348             d = self.bolt.bolt_diameter_provided
349
350             lj_req = Math(inline=True)
351             lj_req.append(NoEscape(r'\begin{aligned}'))
352
353             if l_j > 15 * d:
354                 beta_lj = 1.075 - (l_j / (200 * d))
355                 beta_lj = max(0.75, min(beta_lj, 1.0))
356                 lj_req.append(NoEscape(r'\text{Since } l_j &>
                        15d\\'))
357                 lj_req.append(NoEscape(r'l_j &= ' + str(l_j) +
                        r' \text{ mm}, \quad 15d = ' + str(15 * d)
                        + r' \text{ mm}\\'))
358                 lj_req.append(NoEscape(r'\beta_{lj} &= 1.075 -
                        \frac{l_j}{200 \cdot d}\\'))
359                 lj_req.append(NoEscape(r'&= 1.075 - \frac{' +
                        str(l_j) + r'}{200 \times ' + str(d) + r'
                        }\\'))
360                 lj_req.append(NoEscape(r'&= ' + f'{1.075 - (l_j
```

```
                    / (200 * d)):.3f}' + r'\\'))
361                 lj_req.append(NoEscape(r'&\text{(but } 0.75 \
                        leq \beta_{lj} \leq 1.0\text{)}\\'))
362                 lj_req.append(NoEscape(r'\beta_{lj} &= ' + f'{
                        beta_lj:.2f}' + r'\\'))
363                 lj_status = ""
364             else:
365                 beta_lj = 1.0
366                 lj_req.append(NoEscape(r'\text{Since } l_j &\
                        leq 15d\\'))
367                 lj_req.append(NoEscape(r'l_j &= ' + str(l_j) +
                        r' \text{ mm}, \quad 15d = ' + str(15 * d)
                        + r' \text{ mm}\\'))
368                 lj_req.append(NoEscape(r'\beta_{lj} &= 1.0 \
                        text{ (No reduction)}\\'))
369                 lj_status = "PASS"
370
371             lj_req.append(NoEscape(r'&[\text{Ref. Cl.
                    10.3.3.1}]'))
372             lj_req.append(NoEscape(r'\end{aligned}'))
373
374             lj_prov = Math(inline=True)
375             lj_prov.append(NoEscape(r'\beta_{lj} = ' + f'{
                    beta_lj:.2f}'))
376
377             self.report_check.append(["Long Joint Factor",
                    lj_req, lj_prov, ''])
378
379             l_g = plate1_thk_raw + plate2_thk_raw
380
381             lg_req = Math(inline=True)
382             lg_req.append(NoEscape(r'\begin{aligned}'))
383
384             if l_g > 5 * d:
385                 beta_lg = (8 * d) / (3 * d + l_g)
386                 beta_lg = min(beta_lg, beta_lj) if beta_lj else
                        beta_lg
387                 lg_req.append(NoEscape(r'\text{Since } l_g &> 5
                        d\\'))
```

```
388                 lg_req.append(NoEscape(r'l_g &= ' + str(l_g) +
                        r' \text{ mm}, \quad 5d = ' + str(5 * d) +
                        r' \text{ mm}\\'))
389                 lg_req.append(NoEscape(r'\beta_{lg} &= \frac{8d
                        }{3d + l_g}\\'))
390                 lg_req.append(NoEscape(r'&= \frac{8 \times ' +
                        str(d) + r'}{3 \times ' + str(d) + r' + ' +
                         str(l_g) + r'}\\'))
391                 lg_req.append(NoEscape(r'&= ' + f'{(8 * d) / (3
                         * d + l_g):.3f}' + r'\\'))
392                 lg_req.append(NoEscape(r'\beta_{lg} &\leq \
                        beta_{lj}\\'))
393                 lg_req.append(NoEscape(r'\beta_{lg} &= ' + f'{
                        beta_lg:.2f}' + r'\\'))
394                 lg_status = ""
395             else:
396                 beta_lg = 1.0
397                 lg_req.append(NoEscape(r'\text{Since } l_g &\
                        leq 5d\\'))
398                 lg_req.append(NoEscape(r'l_g &= ' + str(l_g) +
                        r' \text{ mm}, \quad 5d = ' + str(5 * d) +
                        r' \text{ mm}\\'))
399                 lg_req.append(NoEscape(r'\beta_{lg} &= 1.0 \
                        text{ (No reduction)}\\'))
400                 lg_status = "PASS"
401
402             lg_req.append(NoEscape(r'&[\text{Ref. Cl.
                    10.3.3.2}]'))
403             lg_req.append(NoEscape(r'\end{aligned}'))
404
405             lg_prov = Math(inline=True)
406             lg_prov.append(NoEscape(r'\beta_{lg} = ' + f'{
                    beta_lg:.2f}'))
407
408             self.report_check.append(["Large Grip Factor",
                    lg_req, lg_prov, ''])
409
410             if self.bolt.bolt_hole_type != "Standard":
411                 hole_req = Math(inline=True)
```

```
412                 hole_req.append(NoEscape(r'\begin{aligned}'))
413                 hole_req.append(NoEscape(r'\text{Hole Type: }'
                        + self.bolt.bolt_hole_type + r'\\'))
414                 if "oversized" in self.bolt.bolt_hole_type.
                        lower() or "short" in self.bolt.
                        bolt_hole_type.lower():
415                     hole_factor = 0.7
416                 else:  # long-slotted
417                     hole_factor = 0.5
418                 hole_req.append(NoEscape(r'\text{Reduction
                        Factor} &= ' + str(hole_factor) + r'\\'))
419                 hole_req.append(NoEscape(r'&[\text{Ref. Cl.
                        10.3.4}]'))
420                 hole_req.append(NoEscape(r'\end{aligned}'))
421                 self.report_check.append(["Hole Type Reduction"
                        , hole_req, "", ""])
422
423             #=====================================
424             # Section 2.3: Detailing Requirements
425             #=====================================
426             self.report_check.append([
427                 "SubSection", "Detailing Requirements", "|p{4cm
                        }|p{5cm}|p{5.5cm}|p{1.5cm}|"
428             ])
429
430             # 2.3.1 Minimum Spacing (Cl. 10.2.2)
431             p_min = as_int(2.5 * bolt_dia_prov, 0)
432             g_min = as_int(2.5 * bolt_dia_prov, 0)
433
434             spacing_req = Math(inline=True)
435             spacing_req.append(NoEscape(r'\begin{aligned}'))
436             spacing_req.append(NoEscape(r'p_{\text{min}} &= 2.5
                    \cdot d\\'))
437             spacing_req.append(NoEscape(r'&= 2.5 \times ' + str
                    (bolt_dia_prov) + r'\\'))
438             spacing_req.append(NoEscape(r'&= ' + str(p_min) + r
                    ' \text{ mm}\\'))
439             spacing_req.append(NoEscape(r'g_{\text{min}} &= 2.5
                    \cdot d\\'))
```

```
440             spacing_req.append(NoEscape(r'&= 2.5 \times ' + str
                    (bolt_dia_prov) + r'\\'))
441             spacing_req.append(NoEscape(r'&= ' + str(g_min) + r
                    ' \text{ mm}\\ \\'))
442             spacing_req.append(NoEscape(r'&[\text{Ref. Cl.
                    10.2.2}]'))
443             spacing_req.append(NoEscape(r'\end{aligned}'))
444
445             spacing_prov = Math(inline=True)
446             spacing_prov.append(NoEscape(r'\begin{aligned}'))
447             spacing_prov.append(NoEscape(r'p_{\text{prov}} &= '
                     + str(pitch) + r' \text{ mm}\\'))
448             if self.rows > 1:
449                 spacing_prov.append(NoEscape(r'g_{\text{prov}}
                        &= ' + str(gauge) + r' \text{ mm}'))
450             else:
451                 spacing_prov.append(NoEscape(r'g_{\text{prov}}
                        &= \text{N/A}'))
452             spacing_prov.append(NoEscape(r'\end{aligned}'))
453
454             if self.rows > 1:
455                 spacing_status = "PASS" if (pitch >= p_min and
                        gauge >= g_min) else "FAIL"
456             else:
457                 spacing_status = "PASS" if (pitch >= p_min)
                        else "FAIL"
458
459             self.report_check.append(["Minimum Spacing",
                    spacing_req, spacing_prov, spacing_status])
460
461             # 2.3.2 Maximum Spacing (Cl. 10.2.3.1)
462             plate_thk_min = min(plate1_thk_raw, plate2_thk_raw)
463             p_max = as_int(min(32 * plate_thk_min, 300), 0)
464             g_max = as_int(min(32 * plate_thk_min, 300), 0)
465
466             max_spacing_req = Math(inline=True)
467             max_spacing_req.append(NoEscape(r'\begin{aligned}')
                    )
468             max_spacing_req.append(NoEscape(r'p_{\text{max}} &=
```

```
            \min(32 \cdot t, 300 \text{ mm})\\'))
469         max_spacing_req.append(NoEscape(r'&= \min(32 \times
                ' + str(plate_thk_min) + r', 300)\\'))
470         max_spacing_req.append(NoEscape(r'&= ' + str(p_max)
                + r' \text{ mm}\\'))
471         max_spacing_req.append(NoEscape(r'g_{\text{max}} &=
                \min(32 \cdot t, 300 \text{ mm})\\'))
472         max_spacing_req.append(NoEscape(r'&= \min(32 \times
                ' + str(plate_thk_min) + r', 300)\\'))
473         max_spacing_req.append(NoEscape(r'&= ' + str(g_max)
                + r' \text{ mm}\\ \\'))
474         max_spacing_req.append(NoEscape(r'&[\text{Ref. Cl.
                10.2.3.1}]'))
475         max_spacing_req.append(NoEscape(r'\end{aligned}'))
476
477         max_spacing_prov = Math(inline=True)
478         max_spacing_prov.append(NoEscape(r'\begin{aligned}'
                ))
479         max_spacing_prov.append(NoEscape(r'p_{\text{prov}}
                &= ' + str(pitch) + r' \text{ mm}\\'))
480         if self.rows > 1:
481             max_spacing_prov.append(NoEscape(r'g_{\text{
                    prov}} &= ' + str(gauge) + r' \text{ mm}'))
482         else:
483             max_spacing_prov.append(NoEscape(r'g_{\text{
                    prov}} &= \text{N/A}'))
484         max_spacing_prov.append(NoEscape(r'\end{aligned}'))
485
486         if self.rows > 1:
487             max_spacing_status = "PASS" if (pitch <= p_max
                    and gauge <= g_max) else "FAIL"
488         else:
489             max_spacing_status = "PASS" if (pitch <= p_max)
                     else "FAIL"
490
491         self.report_check.append(["Maximum Spacing",
                max_spacing_req, max_spacing_prov,
                max_spacing_status])
492
```

```
493             # 2.3.3 Edge Distance (Cl. 10.2.4)
494             bolt_hole_type_str = str(self.bolt.bolt_hole_type)
                    if hasattr(self.bolt, 'bolt_hole_type') else "
                    Standard"
495             d_hole = IS800_2007.cl_10_2_1_bolt_hole_size(d,
                    bolt_hole_type_str)
496
497             if "Sheared" in edge_type or "hand flame cut" in
                    edge_type:
498                 e_min_calc = f2(1.7 * d_hole, 0.0)
499                 e_min_multiplier = 1.7
500             else:  # Rolled, machine-flame cut, sawn and planed
501                 e_min_calc = f2(1.5 * d_hole, 0.0)
502                 e_min_multiplier = 1.5
503
504             epsilon = math.sqrt(250 / fy)
505             e_max_calc = f2(12 * plate_thk_min * epsilon, 0.0)
506
507             edge_req = Math(inline=True)
508             edge_req.append(NoEscape(r'\begin{aligned}'))
509             edge_req.append(NoEscape(r'e_{\min} &= ' + str(
                    e_min_multiplier) + r' \cdot d_0\\'))
510             edge_req.append(NoEscape(r'&= ' + str(
                    e_min_multiplier) + r' \times ' + f'{d_hole:.1f
                    }' + r' = ' + f'{e_min_calc:.1f}' + r' \text{
                    mm}\\'))
511             edge_req.append(NoEscape(r'e_{\text{max}} &= 12 \
                    cdot t \cdot \varepsilon\\'))
512             edge_req.append(NoEscape(r'&= 12 \times ' + str(
                    plate_thk_min) + r' \times ' + f'{epsilon:.2f}'
                     + r'\\'))
513             edge_req.append(NoEscape(r'&= ' + str(e_max_calc) +
                     r' \text{ mm}\\ \\'))
514             edge_req.append(NoEscape(r'&[\text{Ref. Cl.
                    10.2.4}]'))
515             edge_req.append(NoEscape(r'\end{aligned}'))
516
517             edge_prov = Math(inline=True)
518             edge_prov.append(NoEscape(r'e_{\text{prov}} = ' +
```

```
                str(e_dist) + r' \text{ mm}'))
519
520             edge_status = "PASS" if (e_dist >= e_min_calc and
                    e_dist <= e_max_calc) else "FAIL"
521             self.report_check.append(["Edge Distance", edge_req
                    , edge_prov, edge_status])
522
523             #================================
524             # Section 2.4: Number of Bolts
525             #================================
526             self.report_check.append([
527                 "SubSection", "Number of Bolts Required", "|p{4
                        cm}|p{5cm}|p{5.5cm}|p{1.5cm}|"
528             ])
529
530             bolts_req_initial = math.ceil(axial_kN /
                    bolt_final_cap) if bolt_final_cap > 0 else 0
531
532             bolts_eq = Math(inline=True)
533             bolts_eq.append(NoEscape(r'\begin{aligned}\\'))
534             bolts_eq.append(NoEscape(r'n &= \frac{P}{V_{db
                    }}\\\\'))
535             bolts_eq.append(NoEscape(r'&= \frac{' + str(
                    axial_kN) + r'}{' + str(bolt_final_cap) + r'
                    }\\\\'))
536             bolts_eq.append(NoEscape(r'&= ' + str(
                    bolts_req_initial) + r' \text{ nos.}\\'))
537             bolts_eq.append(NoEscape(r'\end{aligned}'))
538
539             self.report_check.append(["Bolts Required", f" {
                    axial_kN:.2f} kN", bolts_eq, ""])
540
541             #================================
542             # Section 2.5: Bolt Arrangement
543             #================================
544             self.report_check.append([
545                 "SubSection", "Bolt Arrangement", "|p{4cm}|p{5
                        cm}|p{5.5cm}|p{1.5cm}|"
546             ])
```

```
547
548             self.report_check.append([
549                 "Bolt Pattern", "2", f"Arrangement: {rows} rows
                        {cols} columns", ""
550             ])
551
552             #================================
553             # Section 2.6: Base Metal Strength
554             #================================
555             self.report_check.append([
556                 "SubSection", "Base Metal Strength", "|p{4cm}|p
                    {5cm}|p{5.5cm}|p{1.5cm}|"
557             ])
558
559             if is_comp:
560                 base_req = Math(inline=True)
561                 base_req.append(NoEscape(r'\begin{aligned}\\'))
562                 base_req.append(NoEscape(r'P_d &= \frac{A_g \
                    cdot f_y}{\gamma_{m0}}\\\\'))
563                 base_req.append(NoEscape(r'&= \frac{' + str(A_g
                    ) + r' \times ' + str(fy) + r'}{1.10}\\\\')
                    )
564                 base_req.append(NoEscape(r'&= ' + f'{
                    base_metal_capacity_kN:.2f}' + r' \text{ kN
                    }\\'))
565                 base_req.append(NoEscape(r'&[\text{Ref. Cl.
                    7.1.2}]'))
566                 base_req.append(NoEscape(r'\end{aligned}'))
567
568                 base_status = "PASS" if base_metal_capacity_kN
                    >= axial_kN else "FAIL"
569                 self.report_check.append(["Plate Tension
                    Capacity", "", base_req, base_status])
570             else:
571                 # 1. Gross Section Yielding
572                 yield_req = Math(inline=True)
573                 yield_req.append(NoEscape(r'\begin{aligned}\\')
                    )
574                 yield_req.append(NoEscape(r'T_{dg} &= \frac{A_g
```

```
                    \cdot f_y}{\gamma_{m0}}\\\\'))
575                 yield_req.append(NoEscape(r'&= \frac{' + str(
                        A_g) + r' \times ' + str(fy) + r'
                        }{1.10}\\\\'))
576                 yield_req.append(NoEscape(r'&= ' + f'{T_dg:.2f}
                        ' + r' \text{ kN}\\'))
577                 yield_req.append(NoEscape(r'&[\text{Ref. Cl.
                        6.2}]'))
578                 yield_req.append(NoEscape(r'\end{aligned}'))
579                 self.report_check.append(["Gross Section Yield"
                        , "", yield_req, ""])
580
581                 # 2. Net Section Rupture
582                 # Back calculate An for display accuracy
583                 # T_dn = 0.9 * An * fu / 1.25 (in kN)
584                 if fu > 0:
585                     An_disp = (T_dn * 1000.0 * 1.25) / (0.9 *
                            fu)
586                 else:
587                     An_disp = 0.0
588
589                 rup_req = Math(inline=True)
590                 rup_req.append(NoEscape(r'\begin{aligned}\\'))
591                 rup_req.append(NoEscape(r'T_{dn} &= \frac{0.9
                        A_n f_u}{\gamma_{m1}}\\'))
592                 rup_req.append(NoEscape(r'&= \frac{0.9 \times '
                        + f'{An_disp:.2f}' + r' \times ' + str(fu)
                        + r'}{1.25}\\'))
593                 rup_req.append(NoEscape(r'&= ' + f'{T_dn:.2f}'
                        + r' \text{ kN}\\'))
594                 rup_req.append(NoEscape(r'&[\text{Ref. Cl.
                        6.3}]'))
595                 rup_req.append(NoEscape(r'\end{aligned}'))
596                 self.report_check.append(["Net Section Rupture"
                        , "", rup_req, ""])
597
598                 # 3. Block Shear (Cl 6.4)
599                 # Recalculate areas for report clarity
600                 # Note: We use the same logic as the solver's
```

```
                    check_base_metal_strength
601                 n_r = self.rows
602                 p = self.final_pitch
603                 g_val = self.final_gauge
604                 e_val = self.final_end_dist
605                 dia_hole = IS800_2007.cl_10_2_1_bolt_hole_size(
                        d, str(self.bolt.bolt_hole_type))
606
607                 t_min = min(plate1_thk_raw, plate2_thk_raw)
608
609                 Avg = t_min * ((n_r - 1) * g_val + e_val)
610                 Avn = t_min * ((n_r - 1) * g_val + e_val - (n_r
                        - 0.5) * dia_hole)
611                 Atg = t_min * e_val
612                 Atn = t_min * (e_val - 0.5 * dia_hole)
613
614                 Tdb1 = (Avg * fy / (math.sqrt(3) * 1.10) + 0.9
                        * Atn * fu / 1.25) / 1000
615                 Tdb2 = (0.9 * Avn * fu / (math.sqrt(3) * 1.25)
                        + Atg * fy / 1.10) / 1000
616                 Tdb = min(Tdb1, Tdb2)
617
618                 block_req = Math(inline=True)
619                 block_req.append(NoEscape(r'\begin{aligned}'))
620                 block_req.append(NoEscape(r'T_{db1} &= \left( \
                        frac{A_{vg} f_y}{\sqrt{3} \gamma_{m0}} + \
                        frac{0.9 A_{tn} f_u}{\gamma_{m1}} \right)\\
                        '))
621                 block_req.append(NoEscape(r'&= \left( \frac{' +
                         f'{Avg:.0f}' + r' \times ' + str(fy) + r'
                        }{\sqrt{3} \times 1.10} + \frac{0.9 \times
                        ' + f'{Atn:.0f}' + r' \times ' + str(fu) +
                        r'}{1.25} \right)\\'))
622                 block_req.append(NoEscape(r'&= ' + f'{Tdb1:.2f}
                        ' + r' \text{ kN}\\\\'))
623                 block_req.append(NoEscape(r'T_{db2} &= \left( \
                        frac{0.9 A_{vn} f_u}{\sqrt{3} \gamma_{m1}}
                        + \frac{A_{tg} f_y}{\gamma_{m0}} \right)\\'
                        ))
```

```
624                 block_req.append(NoEscape(r'&= \left( \frac{0.9
                        \times ' + f'{Avn:.0f}' + r' \times ' +
                        str(fu) + r'}{\sqrt{3} \times 1.25} + \frac
                        {' + f'{Atg:.0f}' + r' \times ' + str(fy) +
                        r'}{1.10} \right)\\'))
625                 block_req.append(NoEscape(r'&= ' + f'{Tdb2:.2f}
                        ' + r' \text{ kN}\\\\'))
626                 block_req.append(NoEscape(r'T_{db} &= \min(T_{
                        db1}, T_{db2}) = ' + f'{Tdb:.2f}' + r' \
                        text{ kN}\\'))
627                 block_req.append(NoEscape(r'&[\text{Ref. Cl.
                        6.4.1}]'))
628                 block_req.append(NoEscape(r'\end{aligned}'))
629                 self.report_check.append(["Block Shear", "",
                        block_req, ""])
630
631                 # Governing Strength
632                 base_req = Math(inline=True)
633                 base_req.append(NoEscape(r'\begin{aligned}'))
634                 base_req.append(NoEscape(r'T_d &= \min(T_{dg},
                        T_{dn}, T_{db})\\'))
635                 base_req.append(NoEscape(r'&= ' + f'{
                        base_metal_capacity_kN:.2f}' + r' \text{ kN
                        }\\'))
636                 base_req.append(NoEscape(r'\end{aligned}'))
637
638                 base_status = "PASS" if base_metal_capacity_kN
                        >= axial_kN else "FAIL"
639                 self.report_check.append(["Plate Tension
                        Capacity", "", base_req, base_status])
640
641             #============================
642             # Section 2.7: Design Summary
643             #============================
644             self.report_check.append([
645                 "SubSection", "Design Summary", "|p{4cm}|p{5cm
                        }|p{5.5cm}|p{1.5cm}|"
646             ])
647
```

```
            bolt_capacity_total = f2(bolt_final_cap * n_bolts,
                0.0)
            bolt_ur = axial_kN / bolt_capacity_total if
                bolt_capacity_total > 0 else 999.0

            plate_ur = axial_kN / base_metal_capacity_kN if
                base_metal_capacity_kN > 0 else 999.0

            # Overall UR is max of both
            overall_ur_val = max(bolt_ur, plate_ur)
            overall_ur = round(overall_ur_val, 3)

            ur_req = Math(inline=True)
            ur_req.append(NoEscape(r'\begin{aligned}\\'))
            ur_req.append(NoEscape(r'\text{Bolt Capacity} &= '
                + str(bolt_capacity_total) + r' \text{ kN}\\'))
            ur_req.append(NoEscape(r'\text{Plate Capacity} &= '
                 + str(base_metal_capacity_kN) + r' \text{ kN
                }\\\\'))
            ur_req.append(NoEscape(r'\text{UR}_{\text{bolt}} &=
                 \frac{' + str(axial_kN) + r'}{' + str(
                bolt_capacity_total) + r'}\\'))
            ur_req.append(NoEscape(r'&= ' + f'{bolt_ur:.3f}' +
                r'\\\\'))
            ur_req.append(NoEscape(r'\text{UR}_{\text{plate}}
                &= \frac{' + str(axial_kN) + r'}{' + str(
                base_metal_capacity_kN) + r'}\\'))
            ur_req.append(NoEscape(r'&= ' + f'{plate_ur:.3f}' +
                 r'\\\\'))
            ur_req.append(NoEscape(r'\text{UR}_{\text{final}}
                &= \max(\text{UR}_{\text{bolt}}, \text{UR}_{\
                text{plate}})\\'))
            ur_req.append(NoEscape(r'&= ' + str(overall_ur) + r
                '\end{aligned}'))

            util_status = "PASS" if overall_ur_val <= 1.0 else
                "FAIL"
            self.report_check.append(["Utilization Ratio", f"{
                axial_kN:.2f} kN", ur_req, util_status])
```

```
670
671             Disp_2d_image = []
672             Disp_3D_image = "/ResourceFiles/images/3d.png"
673             rel_path = os.path.abspath(".").replace("\\", "/")
674             fname_no_ext = popup_summary.get("filename", "
                    LapJointBoltedReport")
675             folder = popup_summary.get('folder', './reports')
676             os.makedirs(folder, exist_ok=True)
677
678             CreateLatex.save_latex(
679                 CreateLatex(), self.report_input, self.
                        report_check,
680                 popup_summary, fname_no_ext, rel_path,
                        Disp_2d_image, Disp_3D_image,
681                 module=self.module
682             )
683
684             self.logger.info(f"Report generated successfully: {
                    fname_no_ext}.pdf")
685             return True
686
687         except Exception as e:
688             print(f"WARNING in save_design(): {e}")
689             import traceback
690             traceback.print_exc()
691             return False
692
693 %-------------------- end code ---------------
```

4.3.1 Explanation of the Code

The `save_design` function acts as the primary interface for generating the final design report. It integrates user inputs, design calculations, and report formatting into a cohesive workflow. The function's operation can be summarized in the following steps:

1. **Data Collection and Initialization:** The function begins by gathering all necessary design parameters from the user interface and internal class

attributes. This includes material properties (e.g., f_u, f_y), geometric data (plate thickness, width), bolt specifications (grade, diameter, type), and applied loads (tensile or compressive forces).

2. **Design Status Validation:** Before proceeding, the function checks the `design_status` flag. If the design has failed (e.g., due to insufficient capacity or geometric constraints), it generates a report highlighting the failure mode. If the design is safe, it proceeds to compile the full successful design details.
3. **Latex Dictionary Construction:** A comprehensive dictionary, `report_input`, is constructed to map Python variables to their corresponding LaTeX display strings. This includes:
   * **Connection Details:** Module name, loads, and material grades.
   * **Design Checks:** Calculated capacities for shear (V_{dsb}), bearing (V_{dpb}), and tension (T_{db}).
   * **Detailing Parameters:** Pitch (p), gauge (g), and edge distances (e).
4. **Dynamic Equation Formatting:** The function utilizes the `pylatex.utils.NoEscape` class to generate dynamic mathematical expressions. This allows the report to display the actual substitution of values into formulas (e.g., "$V_{dsb} = \frac{400 \times 245}{1.25} = 78.4$ kN") rather than just static text, providing transparency in the calculations.
5. **File Management and Compilation:** Finally, the function handles file system operations using the `os` module to create the required directories. It then invokes the `CreateLatex.save_latex` method to compile the inputs into a professional PDF document, complete with 3D images and proper formatting.

4.4 Documentation

The directory structure used for the Report Generation is organized as follows:

`src/osdag_core`

```
design_report/
    reportGenerator_latex.py

design_type/
    connection/
        butt_joint_bolted.py
Report_functions.py
```

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

```
$ python -m osdag_gui
```

4.5 References

* Osdag Github repository: https://osdag.github.io/
* IS 800:2007, "General Construction in Steel Code of Practice of Practice"

Chapter 5

Report Generation of Welded Lap Joint

5.1 Problem Statement

The main objective was to generate a detailed design report for a **Welded Lap Joint Connection**. Using Osdag's modular structure, this tool automates the design checks, calculation steps, and capacity verifications for fillet-welded lap joints, strictly adhering to IS 800:2007.

5.2 Tasks Done

5.2.1 Development of `save_design()` Function

- **Lead Function for Report Generation:** Constructed the `save_design()` function as the core orchestrator taking user input, performing verifications, and exporting data to LaTeX.
- **Dynamic Data Collection:** Parameters such as plate thicknesses, plate width, material grade, weld size, and load are retrieved from the interface and stored systematically.
- **Comprehensive Calculations:** The function runs mandatory design checks, including:
- **Weld Size Limit Check:** Ensures the weld lies within code-prescribed

minimum and maximum limits, which depend on the thinner connected plate.

- **Weld Strength:**

$$f_{\text{wd}} = \frac{f_u \cdot a}{\sqrt{3}\gamma_{\text{mw}}}$$

where f_u is the weld/filler material strength, a is the effective throat thickness, and γ_{mw} is the weld partial safety factor.

- **Weld Length:** Required effective weld length calculated as

$$l_{\text{eff}} = \frac{\text{Applied Force}}{2f_{\text{wd}}}$$

- **Long Joint Reduction:** Reduces weld capacity for joints exceeding code-specified lengths.
- **Base Metal Strength:** Checks both gross section yielding and net section rupture as per IS 800 (Cl. 6.2, 6.3), including a shear lag factor for lap joints.
- **Utilization Ratio:**

$$UR = \frac{\text{Applied Load}}{\min(\text{Weld Capacity}, \text{Base Metal Capacity})}$$

- **Automated LaTeX Integration:** Results, equations, pass/fail results, detailing, and summary tables are mapped into a LaTeX template for PDF report generation using Osdag's `CreateLatex` system.

5.2.2 Check Implementation and Output

- **Logical Status Verification:** If any check fails (e.g., weld overstressed, length out of code bounds), the report highlights the cause with clear "Fail" status.
- **Summary Table Output:** The report summarizes all key checks (weld, base metal, utilization ratio) in a structured table for audit and review.
- **Deterministic and Reproducible:** The approach ensures consistency of results and easy traceability, fulfilling code and project doc-

umentation requirements.

5.3 Python Code

The entire logic for the report generation of Welded Lap Joint module is within the save_design() function. And it's being connected to the main GUI through ui_template.py and ui_popup_summary.py files. Below are key snippets from the module with explanations of their functionality.

Description of the Script

The script defines the save_design() function, which handles everything from user input to final design report generation. It uses reportGenerator_latex.py to manage properties and perform calculations.

Python Code

The following snippets showcase the core logic implemented in the report generation task.

Listing 5.1: Welded Lap Joint save_design() function

```
1  %--------------------begin code-------------
2  def save_design(self, popup_summary):
3          """
4          Generate the LaTeX design report for Lap Joint
               Welded Connection (Tension/Compression)
5          per IS 800:2007.
6          """
7          try:
8              #=======================================
9              #=========== HELPER FUNCTIONS ==========
10             #=======================================
11             def g(attr, default=None):
12                 """Get attribute value with default"""
13                 v = getattr(self, attr, default)
14                 return default if v is None else v
15
16             def f2(x, default=0.0):
```

```
17                  """Format to 2 decimal places"""
18                  try:
19                      return round(float(x), 2)
20                  except (TypeError, ValueError):
21                      return default
22
23              #
                    ================================================
24              #=========== EXTRACT ALL DESIGN VALUES
                    ==========
25              #
                    ================================================
26              module = g('module', 'Lap Joint Welded')
27              mainmodule = 'Plate(d) Connection'
28              design_for = g('design_for', 'Tension').strip()
29              is_comp = design_for.lower().startswith('c')
30
31              edge_type = g('edgetype', 'Sheared or hand
                    flame cut')
32
33              # Plate properties
34              plate1_thk = f2(self.plate1.thickness[0] if
                    isinstance(self.plate1.thickness, list) else
                     self.plate1.thickness, 0.0)
35              plate2_thk = f2(self.plate2.thickness[0] if
                    isinstance(self.plate2.thickness, list) else
                     self.plate2.thickness, 0.0)
36              plate_thk_min = min(plate1_thk, plate2_thk)
37              width = f2(g('width', 0.0), 0.0)
38              fy = f2(self.plate1.fy if hasattr(self, 'plate1
                    ') else 250, 250)
39              fu = f2(self.plate1.fu if hasattr(self, 'plate1
                    ') else 410, 410)
40
41              # Use stored values for strength calculation
                    variables to match Dock exactly
42              fu_weld = g('fu_weld', float(g('weld.fu', 410))
```

```
            )
        fu_parent = g('fu_parent', min(self.plate1.fu,
            self.plate2.fu) if hasattr(self, 'plate1')
            else 410)

        # Load
        axial_force_N = f2(g('axial_force', g('
            axialforce', g('tensileforce', 0.0))), 0.0)
        axial_kN = f2(axial_force_N / 1000, 0.0)

        # Weld properties
        weld_size = f2(g('weld_size', g('weldsize',
            0.0)), 0.0)
        weld_type = g('weld.type', 'Shop weld')
        weld_fabrication = g('weld.fabrication', 'Shop
            Weld')
        gamma_mw = 1.25 if 'shop' in weld_type.lower()
            else 1.50

        # Effective throat thickness
        effective_throat = f2(g('
            effective_throat_thickness', 0.7 * weld_size
            ), 0.0)

        # Weld lengths
        l_eff = f2(g('l_eff', g('weld_length_effective'
            , g('weldlengtheffective', 0.0))), 0.0)
        l_eff_min = f2(max(4 * weld_size, 40), 0.0)
        l_eff_max = f2(70 * weld_size, 0.0)

        # Long joint reduction factor
        beta_lw = f2(g('beta_lw', g('betalw', 1.0)),
            1.0)

        # End return and connection length
        end_return = f2(g('end_return_length', max(2 *
            weld_size, 12)), 0.0)

        # Overlap: Use calculated value if available to
```

```
            capture clearance logic
70          min_overlap_req = max(4 * weld_size, 40)
71          overlap_length = f2(g('overlap_length',
                min_overlap_req), 0.0)
72
73          conn_length = f2(g('connection_length', l_eff +
                2 * end_return), 0.0)
74
75          # Base metal capacity
76          Ag = plate_thk_min * width
77          gamma_m0 = 1.10
78          gamma_m1 = 1.25
79
80          if is_comp:
81              base_metal_capacity_kN = f2((Ag * fy /
                    gamma_m0) / 1000, 0.0)
82          else:
83              Tdg = (Ag * fy / gamma_m0) / 1000
84              Tdn = (0.9 * Ag * fu * 0.7 / gamma_m1) /
                    1000
85              base_metal_capacity_kN = f2(min(Tdg, Tdn),
                    0.0)
86
87          # Retrieve calculated unit design strengths to
                match Dock
88          P_wd_val = g('weld_design_strength', (fu_weld *
                effective_throat) / (math.sqrt(3) *
                gamma_mw))
89          P_md_val = g('parent_design_strength', 0.6 *
                fu_parent * effective_throat / gamma_mw)
90          P_d_val = g('fillet_weld_design_strength', min(
                P_wd_val, P_md_val))
91
92          #
                ================================================
93          #=========== BUILD REPORT INPUT DICTIONARY
                ==========
94          #
```

```
            ================================================

        self.report_input = {
            KEY_MODULE: module,
            KEY_MAIN_MODULE: mainmodule,
            KEY_DISP_DESIGN_FOR: design_for,
            f"Thickness of Plate-1 (mm) *": plate1_thk,
            f"Thickness of Plate-2 (mm) *": plate2_thk,
            KEY_DISP_PLATE_WIDTH: width,
            KEY_DISP_MATERIAL: g('main_material', g('
                mainmaterial', 'N/A')),
            KEY_DISP_WELD_SIZE: weld_size,
            f"{'Tensile' if not is_comp else 'Axial'}
                Force (kN) *": axial_kN,

            "Additional inputs": "TITLE",
            "Edge Preparation Method": edge_type,
            KEY_DISP_DP_WELD_TYPE: weld_fabrication,
            KEY_DISP_DP_WELD_MATERIAL_G_O_REPORT: f2(
                fu_weld, 410),
        }

        # ========== BUILD REPORT CHECK ==========
        self.report_check = []

        #
            ============================================================

        # SECTION 3.1: CALCULATING WELD STRENGTH
        #
            ============================================================

        # 3.2.1 Weld Size Requirements
        self.report_check.append([
            "SubSection", "Weld Size Check", "|p{4cm}|p
                {6cm}|p{4.5cm}|p{1.5cm}|"
        ])

        s_min = IS800_2007.cl_10_5_2_3_min_weld_size(
```

```
                plate1_thk, plate2_thk)
            if plate_thk_min >= 10:
                s_max = plate_thk_min - 1.5
            else:
                s_max = plate_thk_min
            s_max = f2(s_max, 0.0)

            size_check_req = Math(inline=True)
            size_check_req.append(NoEscape(r'\begin{aligned}'))
            size_check_req.append(NoEscape(r's_{\text{min}} &= ' + str(s_min) + r' \text{ mm}\\'))
            size_check_req.append(NoEscape(r'&[\text{As per Table 21, IS 800:2007}]\\'))
            size_check_req.append(NoEscape(r's_{\text{max}} &= ' + str(s_max) + r' \text{ mm}\\'))
            size_check_req.append(NoEscape(r'&[\text{Ref. Cl. 10.5.2.3, 10.5.2.4}]'))
            size_check_req.append(NoEscape(r'\end{aligned}'))

            size_check_prov = Math(inline=True)
            size_check_prov.append(NoEscape(r's = ' + str(weld_size) + r' \text{ mm}'))

            size_status = "PASS" if (s_min <= weld_size <= s_max) else "FAIL"
            self.report_check.append(["Weld Size", size_check_req, size_check_prov, size_status])

            # 3.1.1 Fillet Weld (Strength Calculation + Throat Thickness)
            self.report_check.append([
                "SubSection", "Weld Strength Calculation", "|p{4cm}|p{6cm}|p{4.5cm}|p{1.5cm}|"
            ])

            # Effective Throat Thickness (Eq 3.2/3.3)
```

```
150             throat_req = Math(inline=True)
151             throat_req.append(NoEscape(r'\begin{aligned}'))
152             throat_req.append(NoEscape(r't_t &= K \times s
                    \\'))
153             throat_req.append(NoEscape(r'&= 0.7 \times ' +
                    str(weld_size) + r'\\'))
154             throat_req.append(NoEscape(r'&= ' + str(
                    effective_throat) + r' \text{ mm}\\'))
155             throat_req.append(NoEscape(r'&[\text{Ref. Cl.
                    10.5.3.2}]'))
156             throat_req.append(NoEscape(r'\end{aligned}'))
157             self.report_check.append(["Throat Thickness", "
                    ", throat_req, ""])
158
159             # Weld Metall Strength (Pwd) (Eq 3.1)
160             weld_metal_req = Math(inline=True)
161             weld_metal_req.append(NoEscape(r'\begin{aligned
                    }\\'))
162             weld_metal_req.append(NoEscape(r'P_{wd} &= \
                    frac{f_u}{\sqrt{3}} \cdot \frac{t_t}{\gamma_
                    {mw}}\\\\'))
163             weld_metal_req.append(NoEscape(r'&= \frac{' +
                    str(f2(fu_weld)) + r'}{\sqrt{3}} \times \
                    frac{' + str(effective_throat) + r'}{' + str
                    (gamma_mw) + r'}\\\\'))
164             weld_metal_req.append(NoEscape(r'&= ' + f'{
                    P_wd_val:.2f}' + r' \text{ N/mm}\\'))
165             weld_metal_req.append(NoEscape(r'&[\text{Ref.
                    Cl. 10.5.7.1.1}]'))
166             weld_metal_req.append(NoEscape(r'\end{aligned}'
                    ))
167             self.report_check.append(["Weld Metal Strength"
                    , "", weld_metal_req, ""])
168
169             # Parent Metal Strength (Pmd) (Eq 3.4)
170             parent_metal_req = Math(inline=True)
171             parent_metal_req.append(NoEscape(r'\begin{
                    aligned}\\'))
172             parent_metal_req.append(NoEscape(r'P_{md} &=
```

```
                0.6 \cdot f_u \cdot \frac{t_t}{\gamma_{mw
                }}\\\\'))
173             parent_metal_req.append(NoEscape(r'&= 0.6 \
                times ' + str(f2(fu_parent)) + r' \times \
                frac{' + str(effective_throat) + r'}{' + str
                (gamma_mw) + r'}\\\\'))
174             parent_metal_req.append(NoEscape(r'&= ' + f'{
                P_md_val:.2f}' + r' \text{ N/mm}\\'))
175             parent_metal_req.append(NoEscape(r'&[\text{Ref.
                 Cl. 10.5.7.1.2}]'))
176             parent_metal_req.append(NoEscape(r'\end{aligned
                }'))
177             self.report_check.append(["Parent Metal
                Strength", "", parent_metal_req, ""])
178
179             # Governing Design Strength (Pd) (Eq 3.5)
180             design_strength_eq = Math(inline=True)
181             design_strength_eq.append(NoEscape(r'\begin{
                aligned}'))
182             design_strength_eq.append(NoEscape(r'P_d &= \
                min(P_{wd}, P_{md})\\'))
183             design_strength_eq.append(NoEscape(r'&= \min('
                + f'{P_wd_val:.2f}' + r', ' + f'{P_md_val:.2
                f}' + r')\\'))
184             design_strength_eq.append(NoEscape(r'&= ' + f'{
                P_d_val:.2f}' + r' \text{ N/mm}\\'))
185             design_strength_eq.append(NoEscape(r'&[\text{
                Ref. Cl. 10.5.7.1}]'))
186             design_strength_eq.append(NoEscape(r'\end{
                aligned}'))
187             self.report_check.append(["Design Strength", ""
                , design_strength_eq, ""])
188
189             # 3.1.2 Reduction Factors (Long Weld)
190             self.report_check.append([
191                 "SubSection", "Long Joint Reduction Factor"
                    , "|p{4cm}|p{6cm}|p{4.5cm}|p{1.5cm}|"
192             ])
193
```

```
            if l_eff > 150 * effective_throat:
                beta_req = Math(inline=True)
                beta_req.append(NoEscape(r'\begin{aligned}'
                    ))
                beta_req.append(NoEscape(r'\text{Since }
                    l_w &> 150 \times t_t\\'))
                beta_req.append(NoEscape(r'\beta_{lw} &=
                    1.2 - 0.2 \times \frac{l_w}{150 \times
                    t_t}\\'))
                beta_req.append(NoEscape(r'&= 1.2 - 0.2 \
                    times \frac{' + str(l_eff) + r'}{150 \
                    times ' + str(effective_throat) + r'}\\'
                    ))
                beta_calc = 1.2 - 0.2 * (l_eff / (150 *
                    effective_throat))
                beta_req.append(NoEscape(r'&= ' + f'{
                    beta_calc:.3f}' + r'\\'))
                beta_req.append(NoEscape(r'&\text{(but }
                    0.6 \leq \beta_{lw} \leq 1.0\text{)}\\')
                    )
                # Use stored beta_lw which should match
                beta_req.append(NoEscape(r'\beta_{lw} &= '
                    + f'{beta_lw:.2f}' + r'\\'))
                beta_req.append(NoEscape(r'&[\text{Ref. Cl.
                     10.5.7.2}]'))
                beta_req.append(NoEscape(r'\end{aligned}'))
                beta_status = ""
            else:
                beta_req = Math(inline=True)
                beta_req.append(NoEscape(r'\begin{aligned}'
                    ))
                beta_req.append(NoEscape(r'\text{Since }
                    l_w &\leq 150 \times t_t\\'))
                beta_req.append(NoEscape(r'\beta_{lw} &=
                    1.0\\'))
                beta_req.append(NoEscape(r'&[\text{Ref. Cl.
                     10.5.7.2}]'))
                beta_req.append(NoEscape(r'\end{aligned}'))
                beta_status = "PASS"
```

```
216
217             beta_prov = Math(inline=True)
218             beta_prov.append(NoEscape(r'\beta_{lw} = ' + f'
                    {beta_lw:.1f}'))
219             self.report_check.append(["Long Joint Factor",
                    beta_req, beta_prov, beta_status])
220
221             #
                    ============================================================
222             # SECTION 3.2: DETAILING CHECKLIST
223             #
                    ============================================================
224
225             # 3.2.2 Effective Length of Weld (Limits)
226             self.report_check.append([
227                 "SubSection", "Effective Length Limits", "|
                        p{4cm}|p{6cm}|p{4.5cm}|p{1.5cm}|"
228             ])
229
230             eff_len_req = Math(inline=True)
231             eff_len_req.append(NoEscape(r'\begin{aligned}')
                    )
232             eff_len_req.append(NoEscape(r'l_{\text{eff,min
                    }} &= \max(4s, 40)\\'))  # Step 1: Formula
233             eff_len_req.append(NoEscape(r'&= \max(4 \times
                    ' + str(weld_size) + r', 40)\\'))  # Step 2:
                     Substitution
234             eff_len_req.append(NoEscape(r'&= ' + str(
                    l_eff_min) + r' \text{ mm}\\'))  # Step 3:
                    Result
235             eff_len_req.append(NoEscape(r'l_{\text{eff,max
                    }} &= 70s\\'))
236             eff_len_req.append(NoEscape(r'&= 70 \times ' +
                    str(weld_size) + r'\\'))
237             eff_len_req.append(NoEscape(r'&= ' + str(
                    l_eff_max) + r' \text{ mm}\\'))
238             eff_len_req.append(NoEscape(r'&[\text{Ref. Cl.
```

```
                10.5.3}]'))
239             eff_len_req.append(NoEscape(r'\end{aligned}'))
240
241             eff_len_prov = Math(inline=True)
242             eff_len_prov.append(NoEscape(r'l_{\text{eff}} =
                    ' + str(l_eff) + r' \text{ mm}'))
243
244             eff_status = "PASS" if (l_eff_min <= l_eff <=
                l_eff_max) else "FAIL"
245             self.report_check.append(["Length Limits",
                eff_len_req, eff_len_prov, eff_status])
246
247             # 3.2.3 End Returns
248             self.report_check.append([
249                 "SubSection", "End Returns", "|p{4cm}|p{6cm
                    }|p{4.5cm}|p{1.5cm}|"
250             ])
251
252             return_req = Math(inline=True)
253             return_req.append(NoEscape(r'\begin{aligned}'))
254             return_req.append(NoEscape(r'\text{Min. Length}
                    &= \max(2s, 12)\\'))
255             return_req.append(NoEscape(r'&= \max(2 \times '
                    + str(weld_size) + r', 12)\\'))
256             return_req.append(NoEscape(r'&= ' + str(
                end_return) + r' \text{ mm}\\'))
257             return_req.append(NoEscape(r'&[\text{Ref. Cl.
                10.5.4.5}]'))
258             return_req.append(NoEscape(r'\end{aligned}'))
259
260             return_prov = Math(inline=True)
261             return_prov.append(NoEscape(str(end_return) + r
                ' \text{ mm}'))
262
263             self.report_check.append(["End Returns",
                return_req, return_prov, "PASS"])
264
265             #
                ================================================================
```

```
266             # SECTION 3.3: DETAILING
267             #
                    ============================================================
268
269             # 3.3.1 Calculating Required Weld Length
270             self.report_check.append([
271                 "SubSection", "Required Weld Length", "|p{4
                        cm}|p{6cm}|p{4.5cm}|p{1.5cm}|"
272             ])
273
274             # Required Length Calculation (Eq 3.15)
275             # Use P_d_val from object to be consistent
276             l_req_base = axial_force_N / (2 * P_d_val) if
                    P_d_val > 0 else 9999
277             l_req_disp = f2(l_req_base, 0.0)
278
279             length_req_calc = Math(inline=True)
280             length_req_calc.append(NoEscape(r'\begin{
                    aligned}\\'))
281             length_req_calc.append(NoEscape(r'l_{\text{req
                    }} &= \frac{P}{2 \cdot P_d}\\\\'))
282             length_req_calc.append(NoEscape(r'&= \frac{' +
                    str(int(axial_force_N)) + r'}{2 \times ' + f
                    '{P_d_val:.2f}' + r'}\\\\'))
283             length_req_calc.append(NoEscape(r'&= ' + str(
                    l_req_disp) + r' \text{ mm}\\\\'))
284             if beta_lw < 1.0:
285                 l_req_final = l_req_base / beta_lw
286                 length_req_calc.append(NoEscape(r'l_{\text{
                        req,mod}} &= \frac{l_{\text{req}}}{\
                        beta_{lw}} = \frac{' + str(l_req_disp) +
                         r'}{' + str(beta_lw) + r'} = ' + f'{
                        l_req_final:.1f}' + r' \text{ mm}\\'))
287             else:
288                 length_req_calc.append(NoEscape(r'&\text{No
                        long joint reduction.}\\'))
289             length_req_calc.append(NoEscape(r'\end{aligned}
```

```
            '))
290
291         self.report_check.append(["Required Length", ""
                , length_req_calc, ""])
292
293         # 3.3.2 Determining Connection Configuration
294         self.report_check.append([
295             "SubSection", "Connection Configuration", "
                    |p{4cm}|p{6cm}|p{4.5cm}|p{1.5cm}|"
296         ])
297
298         config_calc = Math(inline=True)
299         config_calc.append(NoEscape(r'\begin{aligned}')
                )
300         config_calc.append(NoEscape(r'L_{\text{conn}}
                &= l_{\text{eff}} + 2 \times l_{\text{return
                }}\\' ))
301         config_calc.append(NoEscape(r'&= ' + str(l_eff)
                 + r' + 2 \times ' + str(end_return) + r'\\'
                ))
302         config_calc.append(NoEscape(r'&= ' + str(
                conn_length) + r' \text{ mm}\\'))
303         config_calc.append(NoEscape(r'\end{aligned}'))
304
305         self.report_check.append(["Configuration", "",
                config_calc, ""])
306
307         #
                ============================================================
308         # ADDITIONAL CHECKS
309         #
                ============================================================
310
311         # Base Metal Strength
312         self.report_check.append([
313             "SubSection", "Base Metal Strength", "|p{4
                    cm}|p{6cm}|p{4.5cm}|p{1.5cm}|"
```

```
            ])

            if is_comp:
                comp_req = Math(inline=True)
                comp_req.append(NoEscape(r'\begin{aligned}\\'))
                comp_req.append(NoEscape(r'P_d &= \frac{A_g \times f_y}{\gamma_{m0}}\\\\'))
                comp_req.append(NoEscape(r'&= \frac{' + f'{Ag:.1f}' + r' \times ' + str(fy) + r'}{' + str(gamma_m0) + r'}\\\\'))
                comp_req.append(NoEscape(r'&= ' + str(base_metal_capacity_kN) + r' \text{ kN}\\'))
                comp_req.append(NoEscape(r'&[\text{Ref. Cl. 7.1.2}]'))
                comp_req.append(NoEscape(r'\end{aligned}'))
                comp_status = "PASS" if base_metal_capacity_kN >= axial_kN else "FAIL"
                self.report_check.append(["Plate Tension Capacity", f"{axial_kN:.2f} kN", comp_req, comp_status])
            else:
                ten_req = Math(inline=True)
                ten_req.append(NoEscape(r'\begin{aligned}'))
                ten_req.append(NoEscape(r'T_{dg} &= \frac{A_g f_y}{\gamma_{m0}} = ' + f'{Tdg:.2f}' + r' \text{ kN}\\'))
                ten_req.append(NoEscape(r'T_{dn} &= \frac{0.9 A_g f_u \beta}{\gamma_{m1}} = ' + f'{Tdn:.2f}' + r' \text{ kN}\\'))
                ten_req.append(NoEscape(r'T_d &= \min(T_{dg}, T_{dn}) = ' + str(base_metal_capacity_kN) + r' \text{ kN}\\'))
                ten_req.append(NoEscape(r'&[\text{Ref. Cl. 6.2, 6.3}]'))
```

```
333                 ten_req.append(NoEscape(r'\end{aligned}'))
334                 ten_status = "PASS" if
                        base_metal_capacity_kN >= axial_kN else
                        "FAIL"
335                 self.report_check.append(["Plate Tension
                        Capacity", f"{axial_kN:.2f} kN", ten_req
                        , ten_status])
336
337             #
                    ============================================================
338             # UTILIZATION RATIO
339             #
                    ============================================================
340             self.report_check.append([
341                 "SubSection", "Utilization Ratio", "|p{4cm
                        }|p{6cm}|p{4.5cm}|p{1.5cm}|"
342             ])
343
344             n = 2
345             # Use stored capacity from design capacity if
                    available to be exact
346             weld_capacity_val = g('design_capacity', 2 *
                    l_eff * P_d_val * beta_lw)
347             weld_capacity_kN = f2(weld_capacity_val / 1000,
                     0.0)
348
349             # Use stored UR if available
350             utilization_ratio_val = g('utilization_ratio',
                    axial_kN / weld_capacity_kN if
                    weld_capacity_kN > 0 else 0.0)
351             utilization_ratio = f2(utilization_ratio_val,
                    0.0)
352
353             util_req = Math(inline=True)
354             util_req.append(NoEscape(r'\begin{aligned}'))
355             util_req.append(NoEscape(r'P_{\text{capacity}}
                    &= n \times l_{\text{eff}} \times P_d \times
```

```
                \beta_{lw}\\'))
356             util_req.append(NoEscape(r'&= ' + str(n) + r' \
                    times ' + str(l_eff) + r' \times ' + f'{
                    P_d_val:.2f}' + r' \times ' + str(beta_lw) +
                     r'\\'))
357             util_req.append(NoEscape(r'&= ' + str(
                    weld_capacity_kN) + r' \text{ kN}\\\\'))
358             util_req.append(NoEscape(r'\text{UR} &= \frac{P
                    }{P_{\text{capacity}}}\\\\'))
359             util_req.append(NoEscape(r'&= \frac{' + str(
                    axial_kN) + r'}{' + str(weld_capacity_kN) +
                    r'}\\\\'))
360             util_req.append(NoEscape(r'&= ' + str(
                    utilization_ratio) + r'\\'))
361             util_req.append(NoEscape(r'\end{aligned}'))
362
363             util_prov = Math(inline=True)
364             util_prov.append(NoEscape(str(utilization_ratio
                    ) + r' \leq 1.0'))
365
366             util_status = "PASS" if utilization_ratio <=
                    1.0 else "FAIL"
367             self.report_check.append(["Utilization", f"{
                    axial_kN:.2f} kN", util_req, util_status])
368
369             #=========================================
370             #========== GENERATE PDF REPORT ==========
371             #=========================================
372             Disp_2d_image = []
373             Disp_3D_image = "/ResourceFiles/images/3d.png"
374             rel_path = os.path.abspath(".").replace("\\", "
                    /")
375             fname_no_ext = popup_summary.get("filename", "
                    LapJointWeldedReport")
376             folder = popup_summary.get('folder', './reports
                    ')
377             os.makedirs(folder, exist_ok=True)
378
379             CreateLatex.save_latex(
```

```
380                 CreateLatex(), self.report_input, self.
                        report_check,
381                 popup_summary, fname_no_ext, rel_path,
                        Disp_2d_image, Disp_3D_image,
382                 module=self.module
383             )
384             self.logger.info(f"Report generated
                    successfully: {fname_no_ext}.pdf")
385             return True
386
387         except Exception as e:
388             print(f"WARNING in save_design(): {e}")
389             import traceback
390             traceback.print_exc()
391             return False
392 %-------------------- end code ---------------
```

5.3.1 Explanation of the Code

The `save_design` function for the welded lap joint is the central routine that converts the completed design into a structured LaTeX report. Its workflow can be summarized as follows:

1. **Collect Design Data:** All relevant input and output values stored in the welded lap joint object are gathered. This includes: plate dimensions (thickness and width), weld size, weld length, material properties (f_y, f_u), applied load, and intermediate results such as weld capacity, base metal capacity, and utilization ratio.
2. **Check Design Status:** The function reads the internal `design_status` flag:
   - If the design has failed (for example, required weld length exceeds available overlap, or $UR > 1.0$), it prepares a brief failure report explaining the governing reason.
   - If the design is safe, it proceeds to create a full, detailed report.
3. **Prepare `report_input` and Check List:** A dictionary is assembled to pass all data to the LaTeX template. Typical items are:

- Connection description and input summary,
- Stepwise weld design: permissible weld size range, design weld strength f_{wd}, required and provided weld length, long-weld reduction (if any),
- Base metal strength checks (gross yielding, net rupture),
- Final utilization ratio and pass/fail remarks.

Along with this, a list of design checks (DDCL-style) is created so that each check can be printed with its status in the report.

4. **Generate LaTeX Equations:** Using LaTeX strings (often wrapped with `NoEscape`), the function formats the main formulas with substituted numerical values, such as:

$$f_{\mathrm{wd}} = \frac{f_u \cdot a}{\sqrt{3}\,\gamma_{\mathrm{mw}}}, \qquad R_{\mathrm{weld}} = 2\,l_{\mathrm{eff}}\,f_{\mathrm{wd}},$$

and the comparison of applied load with weld and plate capacities. This makes the report transparent and easy to follow.

5. **File Handling and Report Creation:** Finally, the function sets up the output folder (creating it if needed) and calls the common `CreateLatex.save_latex` method. This compiles the LaTeX document into a professional PDF that contains: input summary, calculation steps, DDCL table, and conclusion for the welded lap joint.

5.4 Documentation

The directory structure used for the Report Generation is organized as follows:

```
src/osdag_core

 design_report/
    reportGenerator_latex.py

 design_type/
```

```
    connection/
        lap_joint_welded.py
Report_functions.py
```

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

```
$ python -m osdag_gui
```

5.5 References

- Osdag Github repository: https://osdag.github.io/
- IS 800:2007, "General Construction in Steel Code of Practice of Practice"

Chapter 6

Report Generation of Welded Butt Joint

6.1 Problem Statement

The objective of this task was to develop an automated design and reporting workflow for a **Welded Butt Joint** connection in Osdag. The module must design the weld and base metal in accordance with IS 800:2007, verify all limit states (weld strength, long joint reduction, base metal yielding and rupture) and generate a clear, stepwise LaTeX report for documentation and checking purposes.

6.2 Tasks Done

6.2.1 Design Automation Workflow

- Implemented the `ButtJointWelded` class as a moment connection module, handling all required inputs: plate thicknesses, plate width, material grade, weld size, cover plate option, and axial force.
- Added UI input, detailing and weld preference functions (e.g. `input_values`, `weld_values`, `detailing_values`) to collect user choices such as shop/-field weld and weld material strength.
- Implemented validation in `func_for_validation` to ensure non-empty

and positive values for key fields (plate width, axial force) before starting the design.

6.2.2 Core Design Functions

- **Initialisation (`set_input_values`):** Sets up plates, weld object, design force (tension or compression), cover plate type (single/double) and calculates required cover plate thickness and packing plate thickness as per IS 800:2007 recommendations.
- **Design of Weld (`design_of_weld`):**
- Selects a valid weld size within IS 800 limits using the minimum and maximum weld size rules.
- Computes effective throat thickness and design weld strength

$$f_w = \frac{f_u}{\sqrt{3}\,\gamma_{mw}}.$$

- Calls subsequent functions for weld length, strength verification, long joint reduction and base metal checks.
- **Weld Length (`weld_length`):**
- Checks that the chosen weld size is between the computed $s_{\min}$ and $s_{\max}$.
- Determines required weld length from the design axial force; if possible a straight weld equal to plate width is used, otherwise skewed welds and possible side welds are calculated.
- Stores the provided and effective weld length and weld angle for later reporting.
- **Weld Strength Verification (`weld_strength_verification`):**
- Ensures effective weld length satisfies the minimum requirement $L_{\text{eff}} \geq 4s$.
- Computes weld strength

$$P_w = N_f \cdot 0.707\, s \cdot L_{\text{eff}} \cdot f_w$$

and the weld utilization ratio $UR_{\text{weld}} = P_{\text{req}}/P_w$.

- **Long Joint Reduction (`long_joint_reduction_factor`):**
- For long welds, calculates reduction factor β_L based on L_{eff} and effective throat thickness, and updates the reduced weld strength and utilization.
- **Base Metal Strength (`check_base_metal_strength`):**
- Computes gross area and net area (equal for welded joints) and evaluates:

$$T_{dy} = \frac{A_g f_y}{\gamma_{m0}}, \quad T_{du} = \frac{0.9 A_n f_u}{\gamma_{m1}}$$

  for tension, or compression capacity for compression design.
- Stores base metal utilization ratio.
- **Final Check (`calculate_final_utilization_ratio`):**
- Takes the maximum of weld and base metal utilization ratios as the overall utilization.
- Sets `design_status` to SAFE if all utilization ratios are ≤ 1.0, otherwise UNSAFE with guidance on which component governs.

6.2.3 Report Generation with `save_design`

- The `save_design` function is called only when `design_status` is SAFE. It gathers all important data (forces, material strengths, plate dimensions, weld size, cover plate type) into a `report_input` dictionary.
- It then constructs a detailed `report_check` list containing:
- Weld size limits ($s_{\min}$, $s_{\max}$) and pass/fail status,
- Weld strength calculation with substituted values,
- Base metal capacity in tension or compression,
- Overall capacity, utilization ratio and final design status.
- Mathematical expressions are wrapped using `NoEscape` so that LaTeX formulas (for example the expression of f_w or P_w) appear correctly in the final PDF.

- Finally, `CreateLatex.save_latex` is invoked to generate the formatted PDF report, including 2D/3D images and a summary table of all checks.

6.3 Python Code

The entire logic for the report generation of Welded Butt Joint module is within the save_design() function. And it's being connected to the main GUI through ui_template.py and ui_popup_summary.py files. Below are key snippets from the module with explanations of their functionality.

Description of the Script

The script defines the save_design() function, which handles everything from user input to final design report generation. It uses reportGenerator_latex.py to manage properties and perform calculations.

Python Code

The following snippets showcase the core logic implemented in the report generation task.

Listing 6.1: Welded Butt Joint save_design() function

```
1  %--------------------begin code-------------
2  def save_design(self, popup_summary):
3          """
4          Generate the LaTeX design report for Welded Butt
               Joint Connection
5          per IS 800:2007 following DDCL structure.
6          """
7          try:
8              # ======================================
9              # =========== HELPER FUNCTIONS ==========
10             # ======================================
11             def g(attr, default=None):
12                 """Get attribute value with default"""
```

```
13                 v = getattr(self, attr, default)
14                 return default if v is None else v
15
16             def f2(x, default=0.0):
17                 """Format to 2 decimal places"""
18                 try:
19                     return round(float(x), 2)
20                 except (TypeError, ValueError):
21                     return default
22
23             #
                   ================================================
24             # =========== EXTRACT ALL DESIGN VALUES
                   ==========
25             #
                   ================================================
26             module = g('module', 'Butt Joint Welded
                   Connection')
27             mainmodule = 'Plate(d) Connection'
28             design_for = g('design_for', 'Tension').
                   strip()
29             is_comp = design_for.lower().startswith('c')
30             edge_type = g('edgetype', 'Sheared or hand
                   flame cut')
31
32             # Plate properties
33             plate1_thk = f2(self.plate1.thickness[0] if
                   isinstance(
34                 self.plate1.thickness, list) else self.
                       plate1.thickness, 0.0)
35             plate2_thk = f2(self.plate2.thickness[0] if
                   isinstance(
36                 self.plate2.thickness, list) else self.
                       plate2.thickness, 0.0)
37             plate_thk_min = min(plate1_thk, plate2_thk)
38             width = f2(g('width', g('plates_width', 0.0)
                   ), 0.0)
```

```
39              fy = f2(self.plate1.fy if hasattr(self, '
                    plate1') else 250, 250)
40              fu = f2(self.plate1.fu if hasattr(self, '
                    plate1') else 410, 410)
41
42              # Use stored values for strength calculation
                     variables
43              fu_weld = g('fu', float(g('weld.fu', 410)))
44
45              # Load
46              axial_force_N = f2(
47                  g('axial_force', g('axialforce', g('
                        tensileforce', 0.0))), 0.0)
48              axial_kN = f2(axial_force_N / 1000, 0.0)
49
50              # Cover plate
51              cover_plate_str = g('cover_plate', g(
52                  'cover_plate_type', 'Single-Cover'))
53              N_f = 2 if "double" in cover_plate_str.lower
                    () else 1
54              cover_thk = f2(g('
                    calculated_cover_plate_thickness', 0.0),
                     0.0)
55              packing_thk = f2(g('packing_plate_thickness'
                    ,
56                                 g('packing_thickness', 0.0)
                                      ), 0.0)
57
58              # Weld properties
59              weld_size = f2(g('weld_size', g('weldsize',
                    0.0)), 0.0)
60              weld_type = g('weld.type', 'Shop weld')
61              weld_fabrication = g('weld.fabrication', '
                    Shop Weld')
62              gamma_mw = 1.25 if 'shop' in weld_type.lower
                    () else 1.50
63
64              # Effective throat thickness
65              effective_throat = f2(
```

```
                g('effective_throat_thickness', 0.707 *
                    weld_size), 0.0)

            # Weld lengths - get from object attributes
            L_req = f2(g('L_req', 0.0), 0.0)
            L_eff_provided = f2(g('weld_length_effective
                ', 0.0), 0.0)
            L_provided_line = f2(
                g('weld_length_provided', 0.0), 0.0)  #
                    Per weld line
            L_eff_min = f2(max(4 * weld_size, 40), 0.0)

            # Skew angle and side welds
            skew_angle = f2(g('weld_angle', 0.0), 0.0)
            side_weld_len = f2(g('side_weld_length',
                0.0), 0.0)

            # Calculate L_target per weld line
            L_target = f2(L_req / N_f if N_f > 0 else
                L_req, 0.0)

            # Long joint reduction factor
            beta_L = f2(g('beta_L', g('betalw', 1.0)),
                1.0)

            # Base metal capacity - FIXED: Now
                calculates both Tdg and Tdn
            Ag = plate_thk_min * width
            gamma_m0 = 1.10
            gamma_m1 = 1.25

            # Always calculate both for tension case
            Tdg = f2((Ag * fy / gamma_m0) / 1000, 0.0)
            Tdn = f2((0.9 * Ag * fu / gamma_m1) / 1000,
                0.0)

            if is_comp:
                base_metal_capacity_kN = Tdg  # Only
                    yielding for compression
```

```
96              else:
97                  base_metal_capacity_kN = f2(min(Tdg, Tdn
                        ), 0.0)
98
99              # Weld strength - should use f_w_adjusted if
                    beta_L applied
100             f_w = fu_weld / (math.sqrt(3) * gamma_mw)
101             f_w_adjusted = f2(f_w * beta_L, 0.0)
102
103             # Weld capacity using adjusted strength
104             weld_strength_N = f2(
105                 f_w_adjusted * effective_throat *
                        L_eff_provided * N_f, 0.0)
106             weld_strength_kN = f2(weld_strength_N /
                    1000, 0.0)
107
108             #
                    ================================================
109             # =========== BUILD REPORT INPUT DICTIONARY
                    ==========
110             #
                    ================================================
111             self.report_input = {
112                 KEY_MODULE: module,
113                 KEY_MAIN_MODULE: mainmodule,
114                 KEY_DISP_DESIGN_FOR: design_for,
115                 f"Thickness of Plate-1 (mm) *":
                        plate1_thk,
116                 f"Thickness of Plate-2 (mm) *":
                        plate2_thk,
117                 KEY_DISP_PLATE_WIDTH: width,
118                 KEY_DISP_MATERIAL: g('main_material', g(
                        'mainmaterial', 'N/A')),
119                 KEY_DISP_COVER_PLT: cover_plate_str,
120                 KEY_DISP_WELD_SIZE: weld_size,
121                 f"{'Tensile' if not is_comp else 'Axial
                        '} Force (kN) *": axial_kN,
```

```
122                 "Additional inputs": "TITLE",
123                 "Edge Preparation Method": edge_type,
124                 KEY_DISP_DP_WELD_TYPE: weld_fabrication,
125                 KEY_DISP_DP_WELD_MATERIAL_G_O_REPORT: f2
                        (fu_weld, 410),
126             }
127
128             # ========== BUILD REPORT CHECK ==========
129             self.report_check = []
130
131             #
                    ================================================
132             # SECTION 3.1: COVER PLATE DESIGN
133             #
                    ================================================
134             self.report_check.append([
135                 "SubSection", "Cover Plate Design", "|p
                        {4cm}|p{4cm}|p{6.5cm}|p{1.5cm}|"
136             ])
137
138             # FIXED ISSUE 1: Correct fraction display
139             if N_f == 2:  # Double cover
140                 tcp_numerator = 9
141                 tcp_denominator = 16
142                 tcp_req = f2((9.0 / 16.0) *
                        plate_thk_min, 0.0)
143             else:  # Single cover
144                 tcp_numerator = 5
145                 tcp_denominator = 8
146                 tcp_req = f2((5.0 / 8.0) * plate_thk_min
                        , 0.0)
147
148             tcp_calc = Math(inline=True)
149             tcp_calc.append(NoEscape(r'\begin{aligned}')
                    )
150             tcp_calc.append(NoEscape(r'T_{cp} &\geq \
                    frac{' + str(
```

```
151                 tcp_numerator) + r'}{' + str(
                        tcp_denominator) + r'} \times T_{min
                        }\\'))
152             tcp_calc.append(NoEscape(r'&\geq \frac{' +
                    str(tcp_numerator) + r'}{' + str(
153                 tcp_denominator) + r'} \times ' + str(
                        plate_thk_min) + r'\\'))
154             tcp_calc.append(
155                 NoEscape(r'&\geq ' + str(tcp_req) + r' \
                        text{ mm}\\'))
156             tcp_calc.append(NoEscape(r'\end{aligned}'))
157
158             tcp_prov = Math(inline=True)
159             tcp_prov.append(
160                 NoEscape(r'T_{cp} = ' + str(cover_thk) +
                        r' \text{ mm}'))
161
162             tcp_status = "PASS" if cover_thk >= tcp_req
                    else "FAIL"
163             self.report_check.append(
164                 ["Cover Plate Thickness", tcp_calc,
                        tcp_prov, tcp_status])
165
166             # Packing plate requirement (if applicable)
167             if abs(plate1_thk - plate2_thk) > 0.001 and
                    N_f == 2:
168                 packing_calc = Math(inline=True)
169                 packing_calc.append(NoEscape(r'\begin{
                        aligned}'))
170                 packing_calc.append(NoEscape(r'T_{pack}
                        &= |t_1 - t_2|\\'))
171                 packing_calc.append(
172                     NoEscape(r'&= |' + str(plate1_thk) +
                            r' - ' + str(plate2_thk) + r'
                           |\\'))
173                 packing_calc.append(
174                     NoEscape(r'&= ' + str(packing_thk) +
                            r' \text{ mm}\\'))
175                 packing_calc.append(NoEscape(r'\end{
```

```
                    aligned}'))
176
177                 packing_prov = Math(inline=True)
178                 packing_prov.append(
179                     NoEscape(str(packing_thk) + r' \text
                        { mm}'))
180
181                 self.report_check.append(
182                     ["Packing Plate", packing_calc,
                        packing_prov, "PASS"])
183
184             #
                ============================================================

185             # SECTION 3.2: WELD DESIGN
186             #
                ============================================================

187             self.report_check.append([
188                 "SubSection", "Weld Design", "|p{4cm}|p
                    {4cm}|p{6.5cm}|p{1.5cm}|"
189             ])
190
191             # Minimum and maximum weld size
192             s_min = IS800_2007.cl_10_5_2_3_min_weld_size
                    (
193                 plate1_thk, plate2_thk)
194             s_max = f2(plate_thk_min - 1.5, 0.0)
195
196             size_check_req = Math(inline=True)
197             size_check_req.append(NoEscape(r'\begin{
                    aligned}'))
198             size_check_req.append(
199                 NoEscape(r's_{min} &= ' + str(s_min) + r
                        ' \text{ mm}\\'))
200             size_check_req.append(
201                 NoEscape(r'&[\text{Table 21, IS
                        800:2007}]\\'))
202             size_check_req.append(NoEscape(r's_{max} &=
```

```
                T_{min} - 1.5\\'))
            size_check_req.append(
                NoEscape(r'&= ' + str(plate_thk_min) + r
                    ' - 1.5\\'))
            size_check_req.append(
                NoEscape(r'&= ' + str(s_max) + r' \text{
                    mm}\\'))
            size_check_req.append(
                NoEscape(r'&[\text{Ref. Cl. 10.5.2.3,
                    10.5.2.4}]'))
            size_check_req.append(NoEscape(r'\end{
                aligned}'))

            size_check_prov = Math(inline=True)
            size_check_prov.append(
                NoEscape(r's = ' + str(weld_size) + r' \
                    text{ mm}'))

            size_status = "PASS" if (s_min <= weld_size
                <= s_max) else "FAIL"
            self.report_check.append(
                ["Weld Size Check", size_check_req,
                    size_check_prov, size_status])

            # Effective throat thickness
            throat_calc = Math(inline=True)
            throat_calc.append(NoEscape(r'\begin{aligned
                }'))
            throat_calc.append(NoEscape(r'a &= 0.707 \
                times s\\'))
            throat_calc.append(
                NoEscape(r'&= 0.707 \times ' + str(
                    weld_size) + r'\\'))
            throat_calc.append(
                NoEscape(r'&= ' + str(effective_throat)
                    + r' \text{ mm}\\'))
            throat_calc.append(NoEscape(r'\end{aligned}'
                ))

```

```
229             self.report_check.append(["Effective Throat"
                    , "", throat_calc, ""])
230
231             # Design strength of weld
232             weld_strength_calc = Math(inline=True)
233             weld_strength_calc.append(NoEscape(r'\begin{
                    aligned}\\'))
234             weld_strength_calc.append(
235                 NoEscape(r'f_{wd} &= \frac{f_u}{\sqrt{3}
                        \times \gamma_{mw}}\\\\'))
236             weld_strength_calc.append(NoEscape(
237                 r'&= \frac{' + str(f2(fu_weld)) + r'}{\
                        sqrt{3} \times ' + str(gamma_mw) + r
                        '}\\\\'))
238             weld_strength_calc.append(
239                 NoEscape(r'&= ' + f'{f_w:.2f}' + r' \
                        text{ N/mm}^2\\'))
240             weld_strength_calc.append(NoEscape(r'\end{
                    aligned}'))
241
242             self.report_check.append(
243                 ["Design Strength", "",
                        weld_strength_calc, ""])
244
245             #
                    ================================================
246             # SECTION 3.3: REQUIRED WELD LENGTH
247             #
                    ================================================
248             self.report_check.append([
249                 "SubSection", "Required Weld Length", "|
                        p{4cm}|p{4cm}|p{6.5cm}|p{1.5cm}|"
250             ])
251
252             # Calculate Lreq in Required column, check
                    condition in Provided column
253             length_req_calc = Math(inline=True)
```

```
254                 length_req_calc.append(NoEscape(r'\begin{
                        aligned}\\'))
255                 length_req_calc.append(
256                     NoEscape(r'L_{req} &= \frac{P_N}{f_{wd}
                            \times a \times n_f}\\\\'))
257                 length_req_calc.append(NoEscape(r'&= \frac{'
                        + str(int(axial_force_N)) + r'}{' +
258                                                 f'{f_w:.2f}' + r' \
                                                        times ' + str(
                                                        effective_throat)
                                                         + r' \times ' +
                                                        str(N_f) + r'
                                                        }\\\\'))
259                 length_req_calc.append(
260                     NoEscape(r'&= ' + str(L_req) + r' \text{
                            mm}\\\\'))
261                 length_req_calc.append(
262                     NoEscape(r'&[\text{Ref. Section 3.3,
                            DDCL}]'))
263                 length_req_calc.append(NoEscape(r'\end{
                        aligned}'))
264
265                 # Condition check in Provided column
266                 length_condition = Math(inline=True)
267                 length_condition.append(NoEscape(r'\begin{
                        aligned}'))
268                 length_condition.append(NoEscape(r'L_{req}
                        &\leq w\\'))
269                 length_condition.append(
270                     NoEscape(r'' + str(L_req) + r' &\leq ' +
                            str(width) + r'\\'))
271                 if L_req <= width:
272                     length_condition.append(
273                         NoEscape(r'&\text{Straight weld is
                                sufficient}'))
274                     req_status = "PASS"
275                 else:
276                     length_condition.append(
277                         NoEscape(r'&\text{Proceed to Section
```

```
                        3.4}'))
278                 req_status = ""
279             length_condition.append(NoEscape(r'\end{
                    aligned}'))
280
281             self.report_check.append(
282                 ["Required Length", length_req_calc,
                        length_condition, req_status])
283
284             #
                    ================================================

285             # SECTION 3.4: WELD LENGTH EXTENSION
286             #
                    ================================================

287             if L_req > width:
288                 self.report_check.append([
289                     "SubSection", "Weld Length Extension
                            ", "|p{4cm}|p{4cm}|p{6.5cm}|p
                            {1.5cm}|"
290                 ])
291
292                 # 3.4.1: Skew Angle Method - show full
                        calculation
293                 skew_calc_detail = Math(inline=True)
294                 skew_calc_detail.append(NoEscape(r'\
                        begin{aligned}'))
295                 skew_calc_detail.append(
296                     NoEscape(r'L_{target} &= \frac{L_{
                            req}}{n_f}\\'))
297                 skew_calc_detail.append(
298                     NoEscape(r'&= \frac{' + str(L_req) +
                             r'}{' + str(N_f) + r'}\\'))
299                 skew_calc_detail.append(
300                     NoEscape(r'&= ' + str(L_target) + r'
                             \text{ mm}\\'))
301                 skew_calc_detail.append(
302                     NoEscape(r'&[\text{Ref. Section
```

```
                        3.4.1, Step 1}]'))
303                 skew_calc_detail.append(NoEscape(r'\end{
                        aligned}'))
304
305                 self.report_check.append(
306                     ["Target Length/Line", "",
                            skew_calc_detail, ""])
307
308                 # Calculate skew angle
309                 if skew_angle > 0:
310                     skew_angle_calc = Math(inline=True)
311                     skew_angle_calc.append(NoEscape(r'\
                            begin{aligned}'))
312                     skew_angle_calc.append(
313                         NoEscape(r'\alpha &= \arctan\
                                left(\frac{L_{target} - w}{2
                                w}\right)\\'))
314                     skew_angle_calc.append(NoEscape(r'&=
                            \arctan\left(\frac{' + str(
315                         L_target) + r' - ' + str(width)
                                + r'}{2 \times ' + str(width
                                ) + r'}\right)\\'))
316                     skew_angle_calc.append(
317                         NoEscape(r'&= ' + f'{skew_angle
                                :.1f}' + r'^\circ\\'))
318
319                     # Check constraints 20
                            60
320                     if skew_angle < 20:
321                         skew_angle_calc.append(NoEscape(
322                             r'&\text{Since } \alpha <
                                    20^\circ, \text{ set } \
                                    alpha = 20^\circ\\'))
323                     elif skew_angle > 60:
324                         skew_angle_calc.append(NoEscape(
325                             r'&\text{Since } \alpha >
                                    60^\circ, \text{ set } \
                                    alpha = 60^\circ\\'))
326                     skew_angle_calc.append(
```

```
                NoEscape(r'&\text{(Proceed to Section 3.4.3)}\\'))

            skew_angle_calc.append(
                NoEscape(r'&[\text{Ref. Section 3.4.1, Steps 3-4}]'))
            skew_angle_calc.append(NoEscape(r'\end{aligned}'))

            self.report_check.append(
                ["Skew Angle", skew_angle_calc, f'{skew_angle:.1f}  ', ""])

            # 3.4.2: Provided weld length per line
            L_prov_line_calc = Math(inline=True)
            L_prov_line_calc.append(NoEscape(r'\begin{aligned}'))
            L_prov_line_calc.append(
                NoEscape(r'L_{provided\_line} &= w + 2w \times \tan(\alpha) \\'))
            L_prov_line_calc.append(NoEscape(r'&= ' + str(width) + r' + 2 \times ' + str(
                width) + r' \times \tan(' + f'{skew_angle:.1f}' + r'^\circ) \\'))
            L_prov_line_calc.append(
                NoEscape(r'&= ' + str(L_provided_line) + r' \text{mm}\\'))
            L_prov_line_calc.append(
                NoEscape(r'&[\text{Ref. Section 3.4.2, Step 1}]'))
            L_prov_line_calc.append(NoEscape(r'\end{aligned}'))

            self.report_check.append(
```

```
                        ["Provided Length/Line", "",
                            L_prov_line_calc, ""])

                # 3.4.3: Side weld extension (if needed)
                if side_weld_len > 0:
                    L_provided_total = L_provided_line *
                        N_f

                    side_calc = Math(inline=True)
                    side_calc.append(NoEscape(r'\begin{
                        aligned}'))
                    side_calc.append(
                        NoEscape(r'L_{side} &= \frac{L_{
                            req} - L_{provided}}{n_f}\\'
                            ))
                    side_calc.append(NoEscape(
                        r'&= \frac{' + str(L_req) + r' -
                             ' + str(L_provided_total) +
                             r'}{' + str(N_f) + r'}\\'))
                    side_calc.append(
                        NoEscape(r'&= ' + str(
                            side_weld_len) + r' \text{
                            mm}\\'))

                    # Minimum return weld
                    min_return = max(2 * weld_size, 10)
                    side_calc.append(NoEscape(
                        r'\text{Min. return} &= \max(2s,
                             10) = ' + str(min_return) +
                             r' \text{ mm}\\'))
                    side_calc.append(NoEscape(r'&[\text{
                        Ref. Cl. 10.5.10.2}]'))
                    side_calc.append(NoEscape(r'\end{
                        aligned}'))

                    self.report_check.append(
                        ["Side Weld Length", side_calc,
                            f'{side_weld_len:.1f} mm', "
                            "])
```

```
374
375             #
                    ================================================
376             # SECTION 3.5: WELD STRENGTH VERIFICATION
377             #
                    ================================================
378             self.report_check.append([
379                 "SubSection", "Weld Strength
                        Verification", "|p{4cm}|p{4cm}|p{6.5
                        cm}|p{1.5cm}|"
380             ])
381
382             # Effective length calculation (Section 3.5,
                    Step 1)
383             eff_len_calc_detail = Math(inline=True)
384             eff_len_calc_detail.append(NoEscape(r'\begin
                    {aligned}'))
385             eff_len_calc_detail.append(
386                 NoEscape(r'L_{eff} &= L_{provided\_line}
                        - 2a\\'))
387             eff_len_calc_detail.append(NoEscape(
388                 r'&= ' + str(L_provided_line) + r' - 2 \
                        times ' + str(effective_throat) + r'
                        \\'))
389             eff_len_calc_detail.append(
390                 NoEscape(r'&= ' + str(L_eff_provided) +
                        r' \text{ mm}\\'))
391             eff_len_calc_detail.append(NoEscape(r'\end{
                    aligned}'))
392
393             self.report_check.append(
394                 ["Effective Length", "",
                        eff_len_calc_detail, ""])
395
396             #
                    ================================================
```

```
            # SECTION 3.6: LONG JOINT REDUCTION FACTOR
            #
                ================================================

            self.report_check.append([
                "SubSection", "Long Joint Reduction
                    Factor", "|p{4cm}|p{4cm}|p{6.5cm}|p
                    {1.5cm}|"
            ])

            # Check condition: Leff > 150a
            if L_eff_provided > 150 * effective_throat:
                beta_req = Math(inline=True)
                beta_req.append(NoEscape(r'\begin{
                    aligned}'))
                beta_req.append(
                    NoEscape(r'\text{Since } L_{eff} &>
                        150 \times a\\'))
                beta_req.append(NoEscape(
                    r'' + str(L_eff_provided) + r' &>
                        150 \times ' + str(
                        effective_throat) + r'\\'))
                beta_req.append(NoEscape(
                    r'' + str(L_eff_provided) + r' &> '
                        + f'{150 * effective_throat:.1f}
                        ' + r'\\'))
                beta_req.append(
                    NoEscape(r'\beta_L &= 1.2 - \frac
                        {0.2 \times L_{eff}}{150 \times
                        a}\\'))
                beta_req.append(NoEscape(r'&= 1.2 - \
                    frac{0.2 \times ' + str(
                    L_eff_provided) + r'}{150 \times ' +
                         str(effective_throat) + r'}\\')
                        )
                beta_calc_val = 1.2 - \
                    (0.2 * L_eff_provided) / (150 *
                        effective_throat)
                beta_req.append(
```

```
                    NoEscape(r'&= ' + f'{beta_calc_val
                        :.3f}' + r'\\'))
                beta_req.append(
                    NoEscape(r'&\text{(Ensure } \beta_L
                        \geq 0.8\text{)}\\'))
                beta_req.append(
                    NoEscape(r'\beta_L &= ' + f'{beta_L
                        :.2f}' + r'\\'))
                beta_req.append(NoEscape(r'&[\text{Ref.
                    Cl. 10.5.7.1(b)}]'))
                beta_req.append(NoEscape(r'\end{aligned}
                    '))
                beta_status = "PASS" if beta_L >= 0.8
                    else "FAIL"
            else:
                beta_req = Math(inline=True)
                beta_req.append(NoEscape(r'\begin{
                    aligned}'))
                beta_req.append(
                    NoEscape(r'\text{Since } L_{eff} &\
                        leq 150 \times a\\'))
                beta_req.append(NoEscape(r'' + str(
                    L_eff_provided) +
                                r' &\leq 150 \times ' +
                                    str(effective_throat
                                    ) + r'\\'))
                beta_req.append(NoEscape(r'' + str(
                    L_eff_provided) +
                                r' &\leq ' + f'{150 *
                                    effective_throat:.1f
                                    }' + r'\\'))
                beta_req.append(NoEscape(r'\beta_L &=
                    1.0\\'))
                beta_req.append(NoEscape(r'&[\text{Ref.
                    Cl. 10.5.7.1(b)}]'))
                beta_req.append(NoEscape(r'\end{aligned}
                    '))
                beta_status = "PASS"

```

```
442             beta_prov = Math(inline=True)
443             beta_prov.append(NoEscape(r'\beta_L = ' + f'
                    {beta_L:.2f}'))
444 
445             self.report_check.append(
446                 ["Reduction Factor", beta_req, beta_prov
                        , beta_status])
447 
448             # Weld capacity with reduction factor (
                    Section 3.6, Equation 3.1)
449             weld_cap_calc = Math(inline=True)
450             weld_cap_calc.append(NoEscape(r'\begin{
                    aligned}\\'))
451             weld_cap_calc.append(
452                 NoEscape(r'C_w &= f_{wd}^{adj} \times a
                        \times L_{eff} \times n_f\\\\'))
453             weld_cap_calc.append(
454                 NoEscape(r'&= (\beta_L \times f_{wd}) \
                        times a \times L_{eff} \times n_f
                        \\\\'))
455             weld_cap_calc.append(NoEscape(r'&= (' + f'{
                    beta_L:.2f}' + r' \times ' + f'{f_w:.2f}
                    ' + r') \times ' + str(
456                 effective_throat) + r' \times ' + str(
                        L_eff_provided) + r' \times ' + str(
                        N_f) + r'\\\\'))
457             weld_cap_calc.append(
458                 NoEscape(r'&= ' + str(weld_strength_kN)
                        + r' \text{ kN}\\'))
459             weld_cap_calc.append(NoEscape(r'\end{aligned
                    }'))
460 
461             self.report_check.append(["Weld Capacity", "
                    ", weld_cap_calc, ""])
462 
463             #
                    ================================================
464             # SECTION 3.7: BASE METAL STRENGTH CHECK
```

```
465             #
                    ================================================
466             self.report_check.append([
467                 "SubSection", "Base Metal Strength Check
                        ", "|p{4cm}|p{4cm}|p{6.5cm}|p{1.5cm
                        }|"
468             ])
469
470             if is_comp:
471                 # For compression - only yielding check
472                 comp_calc_req = Math(inline=True)
473                 comp_calc_req.append(NoEscape(r'\begin{
                        aligned}\\'))
474                 comp_calc_req.append(
475                     NoEscape(r'T_{db} &= \frac{A_g \
                            times f_y}{\gamma_{m0}}\\'))
476                 comp_calc_req.append(NoEscape(
477                     r'T_{db} &= \frac{' + f'{Ag:.1f}' +
                            r' \times ' + str(fy) + r'}{' +
                            str(gamma_m0) + r'}\\'))
478                 comp_calc_req.append(
479                     NoEscape(r'&= ' + str(
                            base_metal_capacity_kN) + r' \
                            text{ kN}\\'))
480                 comp_calc_req.append(
481                     NoEscape(r'&[\text{Ref. Section 3.7,
                             Cl. 7.1.2}]'))
482                 comp_calc_req.append(NoEscape(r'\end{
                        aligned}'))
483
484                 comp_status = "PASS" if
                        base_metal_capacity_kN >= axial_kN
                        else "FAIL"
485                 self.report_check.append(
486                     ["Base Metal Capacity", axial_kN,
                            comp_calc_req, comp_status])
487             else:
488                 # For tension - show both calculations
```

```
                    and take min
489                 ten_calc_req = Math(inline=True)
490                 ten_calc_req.append(NoEscape(r'\begin{
                        aligned}\\'))
491                 ten_calc_req.append(NoEscape(
492                     r'T_{db} &= \min\left(\frac{A_g f_y
                            }{\gamma_{m0}}, \frac{0.9 A_n
                            f_u}{\gamma_{m1}}\right)\\'))
493                 ten_calc_req.append(NoEscape(r'&= \min\
                        left(\frac{' + f'{Ag:.1f}' + r' \
                        times ' + str(fy) + r'}{' + str(
494                         gamma_m0) + r'}, \frac{0.9 \times '
                            + f'{Ag:.1f}' + r' \times ' +
                            str(fu) + r'}{' + str(gamma_m1)
                            + r'}\right)\\'))
495                 ten_calc_req.append(
496                     NoEscape(r'&= \min(' + str(Tdg) + r'
                            , ' + str(Tdn) + r')\\'))
497                 ten_calc_req.append(
498                     NoEscape(r'&= ' + str(
                            base_metal_capacity_kN) + r' \
                            text{ kN}\\'))
499                 ten_calc_req.append(
500                     NoEscape(r'&[\text{Ref. Section 3.7,
                             Cl. 6.2, 6.3}]'))
501                 ten_calc_req.append(NoEscape(r'\end{
                        aligned}'))
502
503                 ten_status = "PASS" if
                        base_metal_capacity_kN >= axial_kN
                        else "FAIL"
504                 self.report_check.append(
505                     ["Base Metal Capacity", axial_kN,
                            ten_calc_req, ten_status])
506
507             #
                    ============================================================

508             # SECTION 3.8: DETAILING CHECKLIST
```

```
509             #
                    ================================================
510             self.report_check.append([
511                 "SubSection", "Detailing Checklist", "|p
                        {4cm}|p{4cm}|p{6.5cm}|p{1.5cm}|"
512             ])
513
514             # Check 1: Minimum effective weld length
515             detail_check_1 = Math(inline=True)
516             detail_check_1.append(NoEscape(r'\begin{
                    aligned}'))
517             detail_check_1.append(NoEscape(r'L_{eff} &\
                    geq 4s\\'))
518             detail_check_1.append(
519                 NoEscape(r'' + str(L_eff_provided) + r'
                        &\geq ' + str(4 * weld_size)))
520             detail_check_1.append(NoEscape(r'\end{
                    aligned}'))
521             detail_status_1 = "PASS" if L_eff_provided
                    >= 4 * weld_size else "FAIL"
522             self.report_check.append(
523                 ["Min. Eff. Length", detail_check_1, "",
                        detail_status_1])
524
525             # Check 2: Minimum weld size
526             detail_check_2 = Math(inline=True)
527             detail_check_2.append(NoEscape(r'\begin{
                    aligned}'))
528             detail_check_2.append(NoEscape(r's &\geq s_{
                    min}\\'))
529             detail_check_2.append(
530                 NoEscape(r'' + str(weld_size) + r' &\geq
                        ' + str(s_min)))
531             detail_check_2.append(NoEscape(r'\end{
                    aligned}'))
532             detail_status_2 = "PASS" if weld_size >=
                    s_min else "FAIL"
533             self.report_check.append(
```

```
534                 ["Min. Weld Size", detail_check_2, "",
                        detail_status_2])
535
536             # Check 3: Maximum weld size
537             detail_check_3 = Math(inline=True)
538             detail_check_3.append(NoEscape(r'\begin{
                    aligned}'))
539             detail_check_3.append(NoEscape(r's &\leq T_{
                    min} - 1.5\\'))
540             detail_check_3.append(
541                 NoEscape(r'' + str(weld_size) + r' &\leq
                        ' + str(s_max)))
542             detail_check_3.append(NoEscape(r'\end{
                    aligned}'))
543             detail_status_3 = "PASS" if weld_size <=
                    s_max else "FAIL"
544             self.report_check.append(
545                 ["Max. Weld Size", detail_check_3, "",
                        detail_status_3])
546
547             # Check 4: Skew angle constraints (if
                    applicable)
548             if skew_angle > 0:
549                 detail_check_4 = Math(inline=True)
550                 detail_check_4.append(NoEscape(r'\begin{
                        aligned}'))
551                 detail_check_4.append(
552                     NoEscape(r'20^\circ &\leq \alpha \
                            leq 60^\circ\\'))
553                 detail_check_4.append(
554                     NoEscape(r'20^\circ &\leq ' + f'{
                            skew_angle:.1f}' + r'^\circ \leq
                             60^\circ'))
555                 detail_check_4.append(NoEscape(r'\end{
                        aligned}'))
556                 detail_status_4 = "PASS" if 20 <=
                        skew_angle <= 60 else "FAIL"
557                 self.report_check.append(
558                     ["Skew Angle Range", detail_check_4,
```

```
                        "", detail_status_4])
559
560             # Check 5: Packing plate requirement
561             if N_f == 2:
562                 if abs(plate1_thk - plate2_thk) > 0.001:
563                     detail_check_5 = NoEscape(
564                         r'\text{Required: } |t_1 - t_2|
                            > 0')
565                     detail_prov_5 = f"Provided: {
                        packing_thk} mm"
566                     detail_status_5 = "PASS"
567                 else:
568                     detail_check_5 = NoEscape(
569                         r'\text{Not required: } t_1 =
                            t_2')
570                     detail_prov_5 = "N/A"
571                     detail_status_5 = "PASS"
572                 self.report_check.append(
573                     ["Packing Plate", detail_check_5,
                        detail_prov_5, detail_status_5])
574
575             #
                ================================================
576             # SECTION 3.9: DETAILING
577             #
                ================================================
578             self.report_check.append([
579                 "SubSection", "Detailing", "|p{4cm}|p{4
                    cm}|p{6.5cm}|p{1.5cm}|"
580             ])
581
582             # Length of connection
583             conn_len_detail = Math(inline=True)
584             conn_len_detail.append(NoEscape(r'\begin{
                aligned}'))
585             conn_len_detail.append(
586                 NoEscape(r'L_{provided} &= n_f \times L_
```

```
                {provided\_line}\\'))
587             conn_len_detail.append(
588                 NoEscape(r'&= ' + str(N_f) + r' \times '
                        + str(L_provided_line) + r'\\'))
589             L_provided_total = f2(N_f * L_provided_line,
                    0.0)
590             conn_len_detail.append(
591                 NoEscape(r'&= ' + str(L_provided_total)
                        + r' \text{ mm}'))
592             conn_len_detail.append(NoEscape(r'\end{
                    aligned}'))
593             self.report_check.append(
594                 ["Connection Length", "",
                        conn_len_detail, ""])
595
596             # Length of cover plate (with clearance)
597             end_clearance = 25  # mm, as per Section 3.8
598             L_cover_plate = f2(L_provided_total + 2 *
                    end_clearance, 0.0)
599             cover_len_detail = Math(inline=True)
600             cover_len_detail.append(NoEscape(r'\begin{
                    aligned}'))
601             cover_len_detail.append(
602                 NoEscape(r'L_{cp} &= L_{provided} + 2 \
                        times (\text{end clearance})\\'))
603             cover_len_detail.append(NoEscape(
604                 r'&= ' + str(L_provided_total) + r' + 2
                        \times ' + str(end_clearance) + r'\\
                        '))
605             cover_len_detail.append(
606                 NoEscape(r'&= ' + str(L_cover_plate) + r
                        ' \text{ mm}'))
607             cover_len_detail.append(NoEscape(r'\end{
                    aligned}'))
608             self.report_check.append(
609                 ["Cover Plate Length", "",
                        cover_len_detail, ""])
610
611             #
```

```
            # ==========================================================

            # UTILIZATION RATIO
            # ==========================================================

            self.report_check.append([
                "SubSection", "Utilization Ratio", "|p{4cm}|p{4cm}|p{6.5cm}|p{1.5cm}|"
            ])

            overall_capacity_kN = min(weld_strength_kN, base_metal_capacity_kN)
            utilization_ratio = f2(axial_kN / overall_capacity_kN if overall_capacity_kN > 0 else 0, 0.0)

            ur_calc = Math(inline=True)
            ur_calc.append(NoEscape(r'\begin{aligned}'))
            ur_calc.append(NoEscape(r'UR &= \frac{P_N}{\min(C_w, T_{db})}\\'))
            ur_calc.append(NoEscape(r'&= \frac{' + str(axial_kN) + r'}{\min(' + str(weld_strength_kN) + r', ' + str(base_metal_capacity_kN) + r')}\\'))
            ur_calc.append(NoEscape(r'&= \frac{' + str(axial_kN) + r'}{' + str(overall_capacity_kN) + r'}\\'))
            ur_calc.append(NoEscape(r'&= ' + str(utilization_ratio) + r'\\'))
            ur_calc.append(NoEscape(r'&[\text{Ref. IS 800:2007}]'))
            ur_calc.append(NoEscape(r'\end{aligned}'))

            ur_status = "PASS" if utilization_ratio <= 1.0 else "FAIL"
            self.report_check.append(
                ["Utilization Ratio", f"{axial_kN:.2f} kN", ur_calc, ur_status])
```

```
636
637             #
                    ================================================
638             # GENERATE LATEX REPORT
639             #
                    ================================================
640             Disp_2d_image = []
641             Disp_3D_image = "/ResourceFiles/images/3d.
                    png"
642             rel_path = os.path.abspath(".").replace("\\"
                    , "/")
643             fname_no_ext = popup_summary.get(
644                 "filename", "ButtJointWeldedReport")
645             folder = popup_summary.get('folder', './
                    reports')
646             os.makedirs(folder, exist_ok=True)
647
648             CreateLatex.save_latex(
649                 CreateLatex(), self.report_input, self.
                        report_check,
650                 popup_summary, fname_no_ext, rel_path,
                        Disp_2d_image, Disp_3D_image,
651                 module=self.module
652             )
653             self.logger.info(
654                 f"Report generated successfully: {
                        fname_no_ext}.pdf")
655             return True
656
657         except Exception as e:
658             print(f"WARNING in save_design(): {e}")
659             import traceback
660             traceback.print_exc()
661             return False
662 %-------------------- end code ---------------
```

6.3.1 Explanation of the Code

The `save_design` function for the welded butt joint is responsible for converting the completed design into a structured LaTeX report. Its main steps are:

1. **Collecting Input and Output Data:** All relevant information stored in the `ButtJointWelded` object is gathered. This includes: plate dimensions and material properties, weld size and type, cover plate details (if any), design axial force, weld and base metal capacities, and final utilization ratios.
2. **Checking Design Status:** The function reads the internal `design_status` flag.
3. If the design is *UNSAFE*, it prepares a short report summarizing the failure reason (for example, "weld capacity required factored load" or "base metal utilization > 1.0").
4. If the design is *SAFE*, it proceeds to create a complete, stepwise report.
5. **Preparing `report_input` and Design Checks:** A dictionary `report_input` is created to pass all numerical values and text labels to the LaTeX template. Typical entries cover:
6. Connection description and input summary,
7. Weld design: permissible weld size range, chosen weld size, design weld strength, required and provided weld length, long-joint reduction factor (if applicable),
8. Base metal strength in tension or compression,
9. Final design strength and overall utilization ratio.

   In parallel, a list of design checks (DDCL-style) is assembled, where each check stores its name, governing formula, numerical substitution and PASS/FAIL status for tabular presentation.
10. **Formatting LaTeX Equations:** Important formulas are written as LaTeX math strings, often wrapped with `NoEscape`, for

example:

$$f_w = \frac{f_u}{\sqrt{3}\,\gamma_{mw}}, \qquad P_{\text{weld}} = A_{\text{weld}}\, f_w,$$

and the comparison of applied force with weld and plate capacities. This allows the report to display both the general equation and the corresponding numerical substitution.

11. **File Handling and PDF Generation:** Finally, the function sets up the output directory (creating it if required), and calls the common `CreateLatex.save_latex` routine. This compiles the LaTeX file into a professional PDF report that includes input summary, detailed calculation steps, design check table and conclusion for the welded butt joint.

6.4 Documentation

The directory structure used for the Report Generation is organized as follows:

```
src/osdag_core


 design_report/
    reportGenerator_latex.py


 design_type/
    connection/
       butt_joint_welded.py
Report_functions.py
```

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

```
$ python -m osdag_gui
```

6.5 References

· Osdag Github repository: https://osdag.github.io/

· IS 800:2007, "General Construction in Steel Code of Practice of Practice"

Chapter 7

Standardization of Report Generation Infrastructure

7.1 Problem Statement

The objective was to refine the global report generation engine (`reportGenerator_la`
to provide a more professional and readable output for all Osdag modules. Key challenges included implementing an adaptive table layout for modules with varying data complexity and ensuring that legal disclaimers regarding designer responsibility are present in every generated document.

7.2 Tasks Done

7.2.1 Global Infrastructure Enhancements

- **Adaptive 2-Column Layout Implementation**
  The engine was updated to automatically detect modules that do not require complex 5-column image-integrated tables. For these modules, a balanced 2-column layout (9cm for Keys, 7.5cm for Values) was implemented to maximize scannability.
- **Standardized Legal Disclaimer (Note Section)**
  A mandatory "Note" section was added to the footer of every report

using a `Tabularx` environment. This section clarifies that while Osdag assists in design, the final structural safety and code compliance remain the responsibility of the individual designer.

- **Enhanced Character Wrapping Logic**

  The text-wrapping logic was recalibrated for the new 2-column format, increasing the character threshold to 65 characters before triggering a multi-row split, ensuring cleaner table aesthetics.

7.3 Python Code (reportGenerator_latex.py)

The following snippets illustrate the global changes applied across all Osdag modules.

Listing 7.1: Standardized Note and Adaptive Layout Logic

```
# Logic for simplified 2-column layout in all modules
else:
    # Use a wider 2-column table for simple modules (
        approx 17cm width)
    with doc.create(LongTable('|p{9cm}|p{7.5cm}|',
        row_height=1.2)) as table:
         table.add_hline()
         for i in uiObj:
             if uiObj[i] == "TITLE":
                 table.add_row((MultiColumn(2, align='|c|
                     ', data=bold(i), ),))
             else:
                 table.add_row((NoEscape(i), uiObj[i]))
             table.add_hline()

# Standardized Note added at the end of all reports
doc.append(pyl.Command('vspace', arguments='10mm'))
with doc.create(Tabularx('|X|', row_height=1.5)) as
    table:
     table.add_hline()
     table.add_row((MultiColumn(1, align='|c|', data=bold
         ('Note')),), color='OsdagGreen')
     table.add_hline()
```

```
    table.add_row([NoEscape(r'The sharing of unabridged
       Osdag design reports is encouraged... The output
        shall be owned by the designer...')])
    table.add_hline()
```

7.4 Documentation

The updated report generation system is organized within the following directory structure:

```
src/osdag/
 design_report/
    reportGenerator_latex.py   # Global engine with Note & Layout upd
```

7.5 References

- Osdag GitHub repository: https://osdag.github.io/
- IS 800:2007, "General Construction in Steel — Code of Practice"

Chapter 8

Conclusions

8.1 Tasks Accomplished

During the course of the screening task and report generation, the following key tasks were accomplished:

- **Developed the design report generation mechanism for Bolted Lap Joint Connections:** Implemented the complete workflow to capture user inputs (plate dimensions, bolt grade, load) and generate a detailed design report.
- Created algorithms to verify shear capacity of bolts in single shear planes.
- Integrated logic for checking minimum and maximum edge distances based on hole type (standard/oversized).
- Automated the block shear failure check for the specific geometry of lap joints.
- **Developed the design report generation mechanism for Bolted Butt Joint Connections:** Successfully implemented the comprehensive `save_design()` mechanism for butt joints, handling complex interaction checks.
- Coded logic for Cover Plate thickness verification ($5/8T_{min}$ vs $9/16T_{min}$) per IS 800:2007.
- Implemented capacity reduction factors for packing plates (β_{pkg}), long

joints (β_{lj}), and large grips (β_{lg}).

- Enabled dynamic mathematical rendering of "Required" vs "Provided" values for transparency.
- **Developed the design report generation mechanism for Welded Lap Joint Connections:** Engineered the reporting logic for fillet welds in lap configurations.
- Implemented checks for effective throat thickness ($0.7 \times s$) and minimum weld size based on plate thickness.
- Developed logic to calculate weld strength per unit length and determine the required effective length.
- Added validation for end returns ($2 \times s$) to ensuring detailing compliance in the final report.
- **Updated the design report generation of Welded Butt Joint Connections:** Refined the existing module to support dual-force scenarios (Compression and Tension).
- Differentiated design strength calculations for Complete Joint Penetration (CJP) groove welds in tension vs. compression.
- Implemented site weld reduction factors where applicable.
- Updated the LaTeX template to dynamically display "Axial Tensile Force" or "Axial Compressive Force" labels based on the input load case.
- **Enhanced Global Report Infrastructure:** Upgraded the core `reportGenerator_latex.py` engine to support adaptive layouts, automatically switching between complex 5-column tables and optimized 2-column layouts based on input data complexity.
- **Implemented Standardized Legal Framework:** Integrated a mandatory "Note" section in the footer of all generated reports to clarify design responsibility, ensuring professional compliance across all Osdag modules.

8.2 Skills Developed

Working on the bolted lap joint, bolted butt joint, welded lap joint, welded butt joint modules, and the global report infrastructure helped in developing a broad set of technical and professional skills. These can be grouped as follows:

1. Advanced Python Programming and Software Engineering

- **Modular and Object-Oriented Design:** Designed separate classes and functions for each connection type, clearly separating responsibilities such as input handling, design calculations, checks, and report generation. This improved understanding of encapsulation, cohesion, and reusability.
- **Adaptive Algorithm Design:** Designed logic to dynamically analyze input dictionaries and select the appropriate table structure (Standard vs. Complex) for the report. This honed skills in writing flexible, context-aware code that handles varying data structures gracefully.
- **Global vs. Modular Architecture:** Learned to modify core infrastructure files (`reportGenerator_latex.py`) to effect global changes (like the Legal Note) while ensuring backward compatibility with individual connection modules.
- **Clean Data Flow and Validation:** Implemented structured methods to collect and validate inputs (plate dimensions, bolt/weld properties, loads), managing default values and providing clear feedback on invalid data.
- **Error Handling and Robustness:** Used conditional checks and try–except patterns to handle runtime issues (file paths, division by zero), improving code resilience.

2. LaTeX-Based Report Generation and Mathematical Communication

- **Dynamic Table Environment Management:** Gained proficiency in programmatically manipulating complex LaTeX environments like `LongTable` and `Tabularx` via Python. Learned to control column widths and row heights dynamically based on data content.
- **Automated Document Styling:** Mastered the use of `PyLaTeX` to inject standardized styling elements (such as `OsdagGreen` color headers and legal footers) consistently across different report types.
- **Dynamic Equation Formatting:** Used tools like `NoEscape` to embed Python variables into LaTeX math expressions, enabling the display of "Required" vs. "Provided" values side-by-side. This is critical for creating verifiable calculation proofs.
- **Clear Presentation of Design Checks:** Structured output so that key checks (shear, bearing, yielding, rupture) appear with equations and status (PASS/FAIL), strengthening technical documentation skills.

3. Structural Design and Code Interpretation Skills

- **Complex Connection Constraints (IS 800:2007):** Deepened understanding of specific code provisions for Butt Joints, specifically the logic for Cover Plate thickness ($5/8T_{min}$ vs $9/16T_{min}$) and the application of Shear Capacity reduction factors (β_{pkg}) for packing plates.
- **Connection Design Logic:** Implemented stepwise design procedures for both Bolted connections (shear/bearing capacity, detailing limits) and Welded connections (weld size limits, effective throat). This strengthened the ability to move from codal equations to executable design logic.
- **Utilization Ratio and Safety Assessment:** Developed a consistent approach for calculating utilization ratios and identifying the

governing failure mode to clearly communicate safety margins.

- **Practical Detailing Awareness:** By coding checks for minimum/-maximum edge distances, pitch, gauge, and weld geometry, gained an appreciation of how theoretical design interacts with fabrication constraints.

4. General Professional and Problem-Solving Skills

- **Algorithmic Thinking:** Broke down complex design procedures into smaller logical steps and implemented them systematically.
- **Systematic Debugging:** Resolved issues related to file I/O permissions and cross-platform path handling when generating PDFs, enhancing troubleshooting skills for desktop application development.
- **Interdisciplinary Integration:** Combined knowledge of structural engineering, programming, and technical writing to deliver a complete tool chain from input to final report.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report			
NAME		ROUSHAN RAJ	
PROJECT		OSDAG	
Internship Work Report		FOSSEE Winter Fellowship 2025	
Week	**Dates (2025-26)**	**Key Tasks Accomplished**	**Hours Worked**
Week 1	Dec 1 – Dec 7	**Bolted Lap Joint Report Generation:** • Implemented save_design() for Bolted Lap Joints. • Added logic for shear capacity and bearing checks. • Validated pitch/gauge/edge distance inputs.	40
Week 2	Dec 8 – Dec 14	**Bolted Lap Joint Refinement:** • Integrated Block Shear Failure checks (Cl 6.4). • Fixed edge case bugs in "Standard" vs "Oversized" holes. • Added dynamic math rendering for utilization ratios.	42
Week 3	Dec 15 – Dec 21	**Welded Lap Joint Module:** • Developed report logic for Fillet Welds in Lap configuration. • Implemented Effective Throat and Length checks. • Validated End Return requirements.	38
Week 4	Dec 22 – Dec 28	**Welded Butt Joint (Groove Welds):** • Updated existing module to handle both Tension & Compression. • Differentiated CJP weld strength for different load cases. • Implemented Site Weld reduction factors.	40

Week 5	Dec 29 – Jan 4	**Bolted Butt Joint (Initial):** • Started save_design() implementation for Bolted Butt Joints. • Coded logic for Single vs. Double Cover Plate thickness. • Setup 5-column layout for complex image integration.	35
Week 6	Jan 5 – Jan 11	**Bolted Butt Joint (Advanced):** • Implemented Capacity Reduction Factors. • Added logic for packing plates > 6mm. • Refined utilization ratio logic for combined checks.	45
Week 7	Jan 12 – Jan 18	**Global Infrastructure Update (Part 1):** • Analyzed reportGenerator_latex.py for layout limitations. • Developed "Adaptive Layout" logic to switch between 5-col and 2-col table • Tested simple modules with new 2-col layout.	40
Week 8	Jan 19 – Jan 25	**Global Infrastructure Update (Part 2):** • Implemented standardized "Legal Note" footer for all reports. • Fixed text-wrapping issues for long strings in 2-col layout. • Validated changes across all 4 connection modules.	42
Week 9	Jan 26 – Jan 31	**Final Documentation & Testing:** • Documented all code changes (Tasks Accomplished). • Final dry-run of all modules to ensure stability. • Prepared Fellowship Report LaTeX content.	36
Total			**358**

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.