FOSSEE Summer Fellowship 2025

On

Development of Customizable Report Generation and Implementation of Report Generation for Butt Joint Bolted Module

Submitted by

Srinivas Raghav V C

3rd Year B.Tech Student, Indian Institute of Information Technology Kottayam

Kerala

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

August 12, 2025

Acknowledgments

I would like to express my sincere gratitude to everyone who supported and guided me throughout the course of this project and my internship.

I am deeply thankful to the entire Osdag team, especially Parth Karia, Ajinkya Dahale, and Ajmal Babu M. S, for their invaluable technical guidance and encouragement during my work on the project.

I wish to acknowledge Prof. Siddhartha Ghosh, Principal Investigator of Osdag and Professor in the Department of Civil Engineering at IIT Bombay, for his vision and leadership in the development of the Osdag project.

I am grateful to Prof. Kannan M. Moudgalya, Principal Investigator of FOSSEE and Professor in the Department of Chemical Engineering at IIT Bombay, for providing the opportunity to participate in the FOSSEE Fellowship and for his constant support.

My sincere thanks to FOSSEE Managers, Usha Viswanathan and Vineeta Parmar, and their entire team for their administrative and organizational support throughout the fellowship.

I gratefully acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for facilitating this project and providing the necessary resources.

I would also like to thank my colleagues and fellow interns for their collaboration, encouragement, and insightful discussions during the internship.

Finally, I extend my appreciation to my college, department, head, and principal for their support and encouragement during my studies and this project.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy**access, equity, and quality**by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock**: Collects and validates user inputs.
- **Output Dock**: Displays design results after validation.
- **CAD Window**: Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log**: Shows errors, warnings, and suggestions based on design checks.

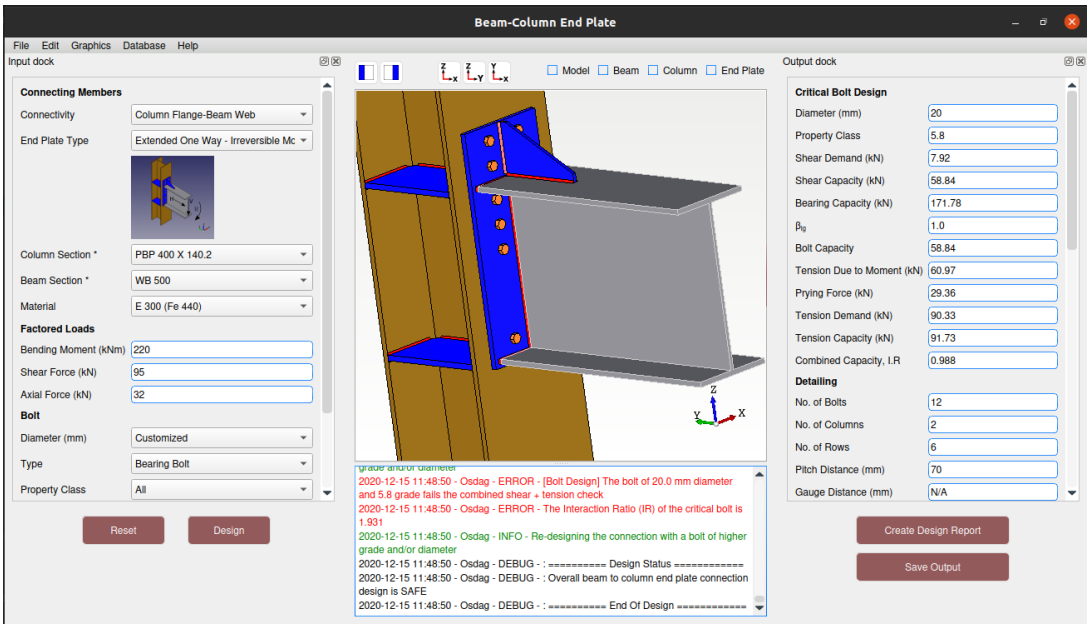


Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model**: The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences**: Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report**: Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Context

This project was undertaken as part of the FOSSEE Fellowship selection process, focusing on the area of Unit Testing and Report Generation. The objective was to demonstrate the ability to automate and customize LaTeX report generation workflows, a critical skill for engineering and scientific software development.

2.2 Problem Statement

In the context of engineering and scientific reporting, it is often necessary to generate customized reports from programmatically created LaTeX files. The challenge addressed in this project was to enable users to select specific components or sections from a LaTeX report (generated using PyLaTeX) and produce a tailored PDF output containing only the desired content. This functionality is essential for improving report relevance, reducing information overload, and supporting diverse stakeholder needs in automated reporting workflows.

2.3 Work Completed

The following objectives were accomplished during the project:

1. **Analysis of PyLaTeX Output:** Examined the structure and content of LaTeX files generated using PyLaTeX to identify logical report components (e.g., sections,

subsections, figures, tables).

2. **Component Extraction Logic:** Developed a parser to identify and extract LaTeX components such as \section{}, \subsection{}, and environments (figures, tables) from the source file.

3. **User Interface for Selection:** Designed a user interface (GUI) that allows users to select which report components to include in the final output.

4. **Custom Report Generation:** Implemented logic to filter the LaTeX source based on user selection and generate a new, customized .tex file.

5. **PDF Compilation:** Automated the compilation of the customized LaTeX file to produce the final PDF report.

6. **Unit Testing:** Developed and executed unit tests to verify the correctness of component extraction, filtering, and report generation.

7. **Documentation:** Documented the workflow, code, and usage instructions for future users and developers.

2.4 Python Code

2.4.1 Description of the Script

The script below parses a LaTeX file, extracts sections and subsections, allows the user to select which components to include, and generates a customized LaTeX file for PDF compilation. The code is modular and designed for extensibility, supporting both command-line and GUI-based workflows.

2.4.2 Implementation

Listing 2.1: Custom LaTeX Report Generator from PyLaTeX Output

```
1 import re
2
3 def extract_sections(latex_content):
4     """
```

```
    Extracts sections and subsections from LaTeX content.
    Returns a dict: {section: [subsections]}
    """
    sections = {}
    current_section = None
    for line in latex_content.split('\n'):
        section_match = re.search(r'\\section\{([^}]+)\}', line)
        if section_match:
            current_section = section_match.group(1)
            sections[current_section] = []
        subsection_match = re.search(r'\\subsection\{([^}]+)\}', line)
        if subsection_match and current_section:
            sections[current_section].append(subsection_match.group(1))
    return sections

def filter_latex(latex_content, selected_sections):
    """
    Filters LaTeX content to include only selected sections/subsections
        .
    """
    lines = latex_content.split('\n')
    filtered = []
    current_section = None
    include = False
    for line in lines:
        section_match = re.search(r'\\section\{([^}]+)\}', line)
        if section_match:
            current_section = section_match.group(1)
            include = current_section in selected_sections
        if include:
            filtered.append(line)
    return '\n'.join(filtered)

# Example usage:
# latex_content = open('input.tex').read()
# sections = extract_sections(latex_content)
# print('Available sections:', sections)
# selected = ['Introduction', 'Results']
# filtered = filter_latex(latex_content, selected)
```

```
# with open('custom_report.tex', 'w') as f:
#     f.write(filtered)
```

2.4.3 Code Overview

- **extract_sections:** Parses the LaTeX file to build a dictionary of sections and their subsections.
- **filter_latex:** Filters the LaTeX content to include only the user-selected sections.
- **Example usage:** Shows how to use the functions to generate a custom report.

2.4.4 Testing and Results

Unit tests were written to verify the correctness of section extraction and filtering. For example, given a LaTeX file with three sections, the script correctly identifies all sections and allows the user to generate a report containing only the selected ones. The output was validated by compiling the filtered LaTeX file to PDF and visually inspecting the result.

2.4.5 Limitations and Future Work

- The current implementation supports only section and subsection extraction. Future work could extend this to include figures, tables, and custom environments.
- The user interface is basic; a more advanced GUI could improve usability and support batch operations.
- Integration with continuous integration (CI) pipelines for automated report testing and generation is a potential area for enhancement.

Chapter 3

Internship Project: Revamp Ideas for Osdag Interface

3.1 Context

This project focused on reimagining the user interface and experience of the Osdag application. The primary objective was to modernize the application's design while maintaining its core functionality and improving user accessibility. The redesign encompassed visual identity, modular architecture, interface design principles, and user experience optimization.

3.2 Problem Statement

The existing Osdag interface required modernization to meet current user experience standards. Key challenges included:

- Outdated visual design that lacked modern appeal
- Inconsistent user interface elements
- Limited accessibility features
- Poor visual hierarchy and navigation
- Lack of cohesive design language

3.3 Design Implementation

3.3.1 Design Philosophy

The redesign was guided by four core principles:

1. Modularity and Scalability

The main idea behind the application was to emphasize modularity, as it greatly enhances maintainability and scalability. By structuring the components in a modular fashion, future updates and feature expansions become significantly more manageable. This architectural approach ensures that:

- Components can be easily modified without affecting others
- New features can be added seamlessly
- Code maintenance becomes more efficient
- Testing and debugging processes are simplified

2. Visual Identity and User Experience

The application consistently follows a green-themed design, providing a unique and recognizable visual identity. This cohesive color scheme reinforces our brand and ensures a seamless user experience across different sections. The green theme was chosen for its:

- Professional appearance suitable for engineering software
- Calming effect that reduces eye strain during long usage
- Association with growth and progress
- High contrast ratios for accessibility

3. Interface Design and Usability

Most buttons feature rounded edges for a modern, friendly appearance. Additionally, the logos have been refined to be more visually clear and accessible. Overall, the interface

has been designed to feel intuitive and user-friendly, ensuring a smoother interaction experience. Key improvements include:

- Rounded button designs for modern aesthetics
- Refined logo designs for better clarity
- Intuitive navigation patterns
- Consistent spacing and typography
- Clear visual hierarchy

4. Dark Mode Implementation

A dark mode palette was implemented as an initial concept, though it requires further refinement. The current implementation provides basic dark mode functionality but needs more creative development for optimal user experience.

3.4 Implementation Details

3.4.1 Modular Component Architecture

The redesign implemented a component-based architecture using PyQt5, where each UI element is treated as a reusable component. This approach provides:

- Consistent behavior across similar elements
- Easy maintenance and updates
- Scalable design system
- Reduced code duplication

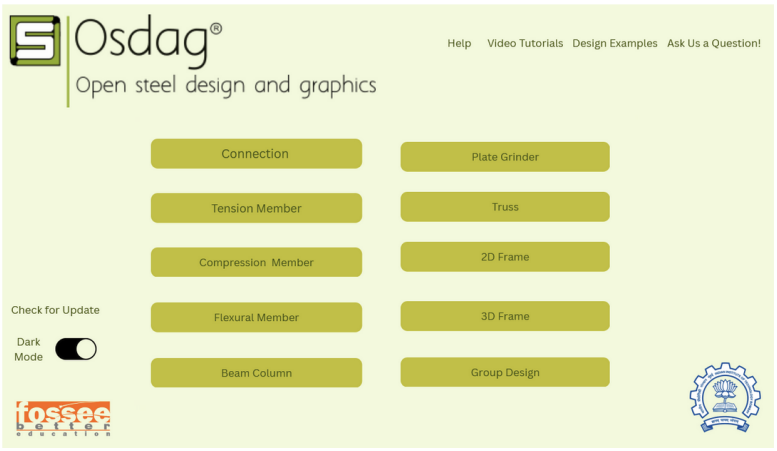
3.4.2 Visual Design System

A comprehensive design system was developed including:

- Color palette with primary green theme

- Typography hierarchy for different content types
- Spacing system for consistent layouts
- Icon set for common actions
- Button and input field styles

3.4.3 User Interface Improvements



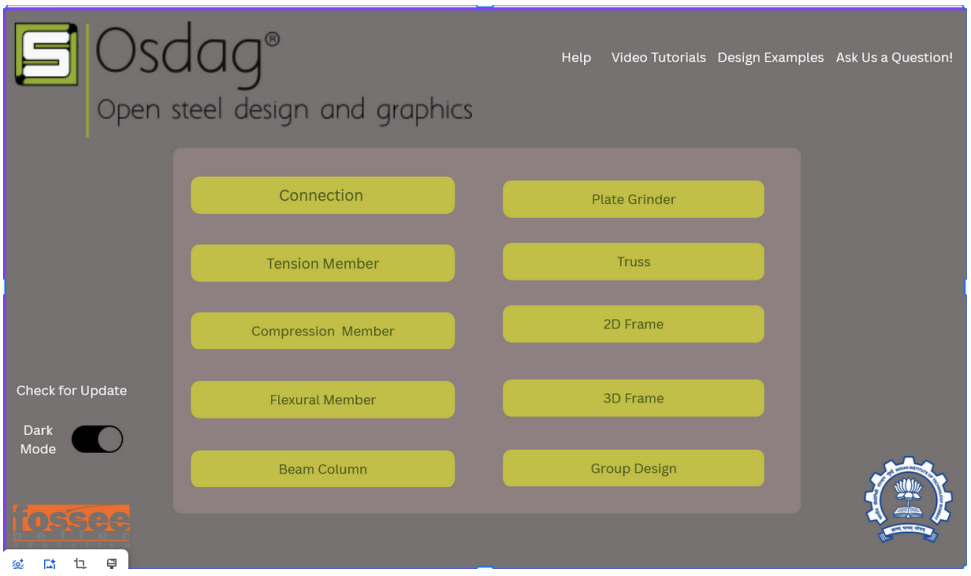


Figure 3.1: Navigation and Menu System - Improved User Flow

3.4.4 Iterative Design Process

The redesign process involved multiple iterations and user feedback cycles:

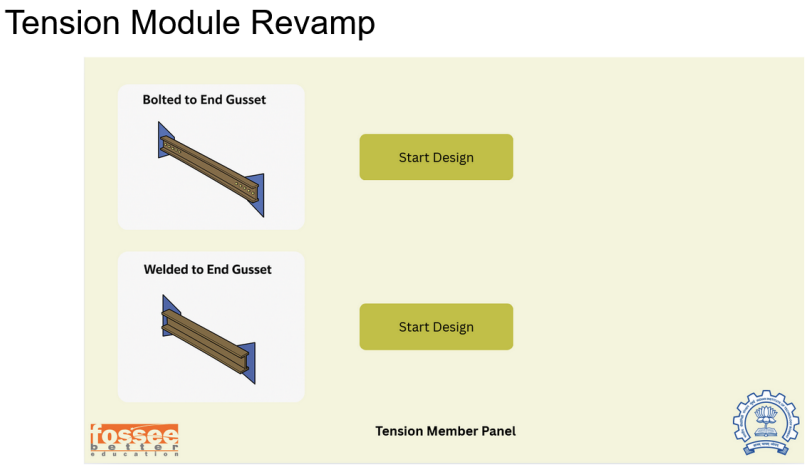
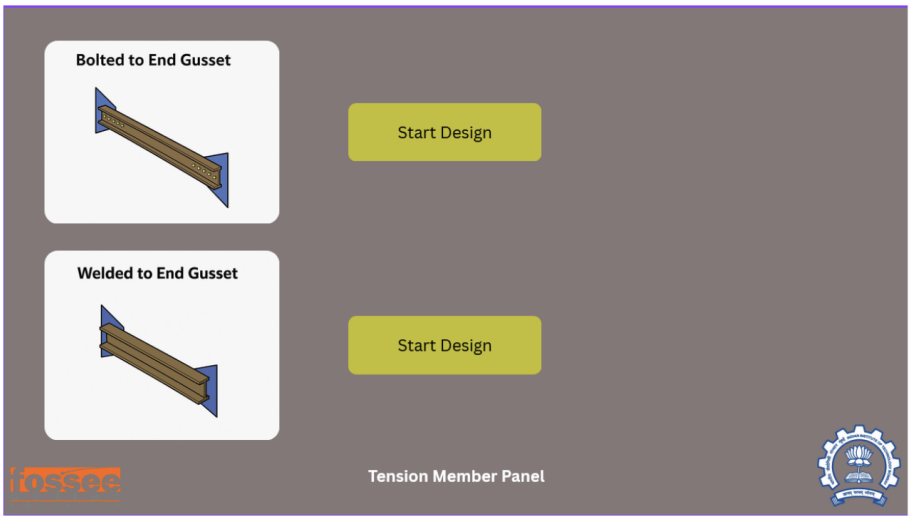


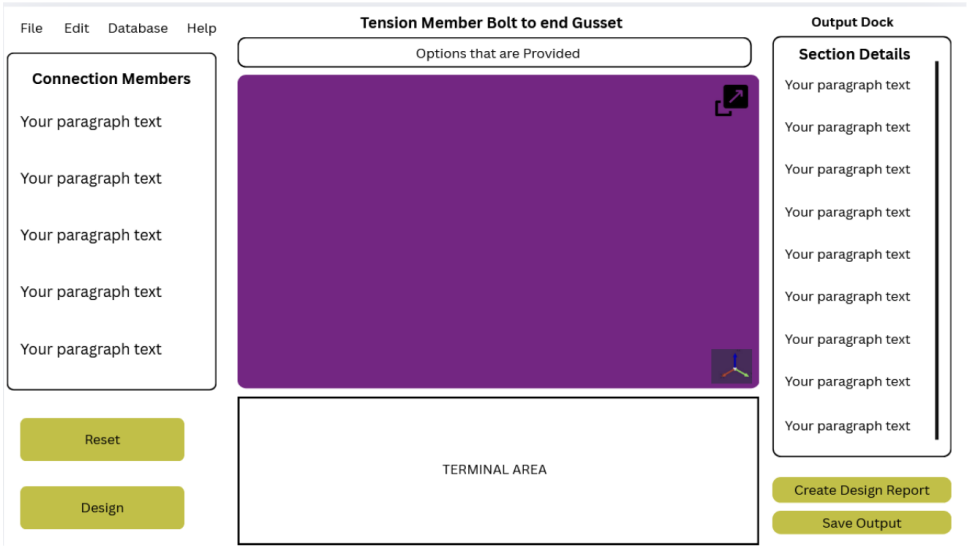
Figure 3.2: Design Iteration Process - User Feedback Integration

3.4.5 Accessibility Enhancements

Special attention was paid to accessibility features:

- High contrast color combinations
- Keyboard navigation support
- Screen reader compatibility
- Scalable text sizes
- Clear focus indicators





3.4.6 Final Implementation

The completed redesign showcases the modern, professional interface:

3.5 Conclusion

The UI/UX redesign project successfully modernized the Osdag application while maintaining its core functionality. The implementation of modular architecture, consistent visual design, and improved user experience has positioned the application for future growth and development. The project demonstrated the importance of user-centered design and the value of iterative development processes.

Chapter 4

Internship Project: Enhancements for Butt Joint Bolted Connection Reporting in Osdag

4.1 Context

This work was carried out as part of the Osdag project to improve the transparency, clarity, and code compliance of bolted connection design reports, specifically for butt joint bolted connections. The enhancements were motivated by the need for explicit, step-by-step documentation of IS 800:2007 reduction factors and robust, user-friendly reporting workflows for both developers and end users.

4.2 Problem Statement

The design and verification of bolted connections in steel structures, such as butt joints, require strict adherence to IS 800:2007. The existing Osdag implementation lacked explicit, step-by-step reporting for long joint and large grip reduction factors, as well as a robust, user-friendly `save_design` function for the Butt Joint Bolted module. This project aimed to implement these missing features, ensuring clarity, transparency, and code compliance in the generated design reports.

4.3 Implementation Overview

1. **Added Functions for Reduction Factors:** Implemented `long_joint_reduction_factor` and `large_grip_reduction_factor` to compute and document the relevant IS 800:2007 reduction factors, with LaTeX output for reporting.

2. **Enhanced save_design Logic:** Developed a comprehensive `save_design` method for the Butt Joint Bolted module, mirroring the reporting style of the tension bolted module and including all relevant checks, calculations, and reporting steps.

3. **Integration and Error Handling:** Ensured robust error handling and minimal error reporting in case of failures, and integrated the new logic with the Osdag reporting workflow.

4. **Testing and Validation:** Verified the correctness of the new functions and reporting logic with various design scenarios, including edge cases and error conditions.

5. **Documentation:** Provided clear code comments, user/developer documentation, and sample outputs for future reference.

4.4 Python Code

4.4.1 Script Architecture

The script introduces two new functions for IS 800:2007 reduction factors and a comprehensive `save_design` method for the Butt Joint Bolted connection. The workflow is as follows:

- **Reduction Factor Functions:** Compute and format the long joint and large grip reduction factors as per IS 800:2007, outputting LaTeX for inclusion in reports.

- **save_design Method:** Gathers all input parameters, performs design checks (spacing, bolt design, reduction factors, verification), and generates a detailed LaTeX report using the Osdag reporting engine.

- **Error Handling:** If any error occurs, a minimal error report is generated and logged.

4.4.2 Source Code

```
    def save_design(self, popup_summary):
        """
        Generate design report for Bolted Butt Joint Connection
           as per IS 800:2007
        Follows tension bolted module reporting style with
           explicit, step-by-step calculations
        """

        try:
            # Build report_input dictionary - Input Parameters
               Section (like tension bolted)
            self.report_input = {
                KEY_MODULE: getattr(self, 'module', 'Butt Joint
                   Bolted'),
                KEY_MAIN_MODULE: getattr(self, 'mainmodule', '
                   Butt Joint Bolted Connection'),

                # Applied Load - Input
                KEY_DISP_TENSILE_FORCE: float(getattr(self, '
                   tensile_force', 0)),

                # Connection Details - Input
                "Connection Details": "TITLE",
                KEY_DISP_MATERIAL: getattr(self, 'main_material',
                    'N/A'),
                KEY_DISP_PLATE1_THICKNESS: float(getattr(self, '
                   plate1thk', 0)),
                KEY_DISP_PLATE2_THICKNESS: float(getattr(self, '
                   plate2thk', 0)),
                KEY_DISP_PLATE_WIDTH: float(getattr(self, 'width'
                   , 0)),
```

```
                KEY_DISP_COVER_PLATE: getattr(self, 'cover_plate'
                    , 'Both Sides'),

                # Material Properties
                "Material Properties": "TITLE",
                KEY_DISP_ULTIMATE_STRENGTH_REPORT: round(getattr(
                    self.plate1, 'fu', 0), 1) if hasattr(self, '
                    plate1') else 0,
                KEY_DISP_YIELD_STRENGTH_REPORT: round(getattr(
                    self.plate1, 'fy', 0), 1) if hasattr(self, '
                    plate1') else 0,

                # Bolt Details - Input and Design Preference (
                    show full lists like tension bolted)
                "Bolt Details - Input and Design Preference": "
                    TITLE",
                KEY_DISP_D: str([int(d) for d in getattr(self.
                    bolt, 'bolt_diameter', [])]) if hasattr(self,
                    'bolt') else "[]",
                KEY_DISP_GRD: str([float(d) for d in getattr(self
                    .bolt, 'bolt_grade', [])]) if hasattr(self, '
                    bolt') else "[]",
                KEY_DISP_TYP: getattr(self.bolt, 'bolt_type', 'N/
                    A') if hasattr(self, 'bolt') else 'N/A',
                KEY_DISP_DP_BOLT_HOLE_TYPE: getattr(self.bolt, '
                    bolt_hole_type', 'Standard') if hasattr(self,
                    'bolt') else 'Standard',

                # Detailing - Design Preference
                "Detailing - Design Preference": "TITLE",
                KEY_DISP_DP_DETAILING_EDGE_TYPE: getattr(self.
                    bolt, 'edge_type', 'Sheared or hand flame cut'
                    ) if hasattr(self, 'bolt') else 'Sheared or
                    hand flame cut',
```

```
            KEY_DISP_DP_DETAILING_CORROSIVE_INFLUENCES_BEAM:
                getattr(self.bolt, 'corrosive_influences', '
                Corrosive') if hasattr(self, 'bolt') else '
                Corrosive',
        }

        # Add bolt-type specific inputs (only if friction
            grip)
        if hasattr(self, 'bolt') and getattr(self.bolt, '
            bolt_type', '') == TYP_FRICTION_GRIP:
            self.report_input[
                KEY_DISP_DP_BOLT_SLIP_FACTOR_REPORT] = getattr
                (self.bolt, 'mu_f', 0.3) if hasattr(self.bolt,
                 'mu_f') else 0.3

        # Build report_check list - Design Verification (like
            tension bolted structure)
        self.report_check = []

        if getattr(self, 'design_status', False):
            # Extract values for calculations
            bolt_diameter_provided = float(getattr(self.bolt,
                 'bolt_diameter_provided', 0)) if hasattr(self
                , 'bolt') else 0.0
            bolt_grade = getattr(self.bolt, '
                bolt_grade_provided', 0) if hasattr(self, '
                bolt') else 0
            bolt_fu = getattr(self.bolt, 'bolt_fu', 0) if
                hasattr(self, 'bolt') else 0
            bolt_fy = getattr(self.bolt, 'bolt_fy', 0) if
                hasattr(self, 'bolt') else 0
            bolt_net_area = getattr(self.bolt, 'bolt_net_area
                ', 0) if hasattr(self, 'bolt') else 0
```

```
                connecting_plates = [getattr(self, 'plate1thk',
                    0), getattr(self, 'plate2thk', 0)]

                # Spacing values
                final_pitch = float(getattr(self, 'final_pitch',
                    0))
                final_gauge = float(getattr(self, 'final_gauge',
                    0))
                final_edge_dist = float(getattr(self, '
                    final_edge_dist', 0))
                final_end_dist = float(getattr(self, '
                    final_end_dist', 0))

                # Bolt layout
                rows = getattr(self, 'rows', 1)
                cols = getattr(self, 'cols', 1)
                number_bolts = int(getattr(self, 'number_bolts',
                    0))
                tensile_force = float(getattr(self, '
                    tensile_force', 0))

                # SECTION 1: Spacing Check (like tension bolted)
                t7 = ('SubSection', 'Spacing Check as per Cl.
                    10.2 of IS 800:2007', '|p{2.5cm}|p{7.5cm}|p{3
                    cm}|p{2.5cm}|')
                self.report_check.append(t7)

                t8 = (DISP_MIN_PITCH, cl_10_2_2_min_spacing(
                    bolt_diameter_provided),
                       display_prov(final_pitch, "p", "mm"),
                       get_pass_fail(2.5 * bolt_diameter_provided,
                           final_pitch, relation='leq'))
                self.report_check.append(t8)
```

```
                t9 = (DISP_MAX_PITCH, cl_10_2_3_1_max_spacing(
                    connecting_plates),
                        display_prov(final_pitch, "p", "mm"),
                        get_pass_fail(final_pitch, 32 * min(
                            connecting_plates), relation='leq'))
                self.report_check.append(t9)

                if final_gauge > 0:  # Only show for multi-row
                    arrangements
                    t10 = (DISP_MIN_GAUGE, cl_10_2_2_min_spacing(
                        bolt_diameter_provided),
                            display_prov(final_gauge, "g", "mm"),
                            get_pass_fail(2.5 *
                                bolt_diameter_provided, final_gauge
                                , relation="leq"))
                    self.report_check.append(t10)

                    t11 = (DISP_MAX_GAUGE,
                        cl_10_2_3_1_max_spacing(connecting_plates)
                        ,
                            display_prov(final_gauge, "g", "mm"),
                            get_pass_fail(final_gauge, 32 * min(
                                connecting_plates), relation="leq")
                                )
                    self.report_check.append(t11)

                edge_type_str = getattr(self.bolt, 'edge_type', '
                    Sheared or hand flame cut') if hasattr(self, '
                    bolt') else 'Sheared or hand flame cut'

                t12 = (DISP_MIN_END,
                    cl_10_2_4_2_min_edge_end_dist(
                    bolt_diameter_provided, edge_type_str),
```

```
                display_prov(final_end_dist, "e_{end}", "
                    mm"),
                get_pass_fail(1.2 * bolt_diameter_provided
                    , final_end_dist, relation='leq'))
            self.report_check.append(t12)

            t13 = (DISP_MIN_EDGE,
                cl_10_2_4_2_min_edge_end_dist(
                bolt_diameter_provided, edge_type_str),
                    display_prov(final_edge_dist, "e", "mm"),
                    get_pass_fail(1.2 * bolt_diameter_provided
                    , final_edge_dist, relation='leq'))
            self.report_check.append(t13)

            # SECTION 3: Bolt Design (like tension bolted)
            t14 = ('SubSection', 'Bolt Design as per Cl. 10.3
                of IS 800:2007', '|p{2.5cm}|p{5.5cm}|p{6.5cm
                }|p{1cm}|')
            self.report_check.append(t14)

            # Bolt capacity calculations
            bolt_type = getattr(self.bolt, 'bolt_type', '')
                if hasattr(self, 'bolt') else ''
            planes = getattr(self, 'planes', 1)
            bolt_shear_capacity = getattr(self.bolt, '
                bolt_shear_capacity', 0) if hasattr(self, '
                bolt') else 0
            bolt_bearing_capacity = getattr(self.bolt, '
                bolt_bearing_capacity', 0) if hasattr(self, '
                bolt') else 0
            bolt_capacity = getattr(self.bolt, 'bolt_capacity
                ', 0) if hasattr(self, 'bolt') else 0

            # Convert to kN for display
```

```
                bolt_shear_capacity_kn = round(
                    bolt_shear_capacity / 1000, 2) if
                    bolt_shear_capacity > 1000 else
                    bolt_shear_capacity
                bolt_bearing_capacity_kn = round(
                    bolt_bearing_capacity / 1000, 2) if
                    bolt_bearing_capacity > 1000 else
                    bolt_bearing_capacity
                bolt_capacity_kn = round(bolt_capacity / 1000, 2)
                    if bolt_capacity > 1000 else bolt_capacity

                if bolt_type == TYP_BEARING:
                    t15 = (KEY_OUT_DISP_BOLT_SHEAR, '',
                           cl_10_3_3_bolt_shear_capacity(bolt_fu,
                               planes, bolt_net_area, 1.25,
                              bolt_shear_capacity_kn), '')
                    self.report_check.append(t15)

                    kb = getattr(self.bolt, 'kb', 0) if hasattr(
                       self, 'bolt') else 0
                    bolt_conn_plates_t_fu_fy = getattr(self, '
                       bolt_conn_plates_t_fu_fy', []) if hasattr(
                       self, 'bolt_conn_plates_t_fu_fy') else []

                    t16 = (KEY_OUT_DISP_BOLT_BEARING, '',
                           cl_10_3_4_bolt_bearing_capacity(kb,
                              bolt_diameter_provided,
                              bolt_conn_plates_t_fu_fy, 1.25,
                              bolt_bearing_capacity_kn), '')
                    self.report_check.append(t16)

                    t17 = (KEY_OUT_DISP_BOLT_CAPACITY, '',
                           cl_10_3_2_bolt_capacity(
                              bolt_shear_capacity_kn,
```

```
                bolt_bearing_capacity_kn,
                bolt_capacity_kn), '')
        self.report_check.append(t17)
    else:
        # HSFG bolt
        mu_f = float(getattr(self.bolt, 'mu_f', 0.3))
            if hasattr(self, 'bolt') else 0.3
        slip_res = getattr(self, 'slip_res', 0)
        slip_res_kn = round(slip_res / 1000, 2) if
           slip_res > 1000 else slip_res
        bolt_capacity_kn = slip_res_kn

        t15 = (KEY_OUT_DISP_BOLT_SLIP_DR, '',
               cl_10_4_3_HSFG_bolt_capacity(mu_f,
                  planes, 1.0, bolt_fu, bolt_net_area
                  , 1.25, slip_res_kn), '')
        self.report_check.append(t15)

    # Number of bolts required
    t18 = (DISP_NUM_OF_BOLTS, '', display_prov(
       number_bolts, "n"), '')
    self.report_check.append(t18)

    # Note: class variables are self.cols (transverse
       ) and self.rows (longitudinal)
    t19 = (DISP_NUM_OF_COLUMNS, '', display_prov(self
       .cols, "n_{c}"), '')
    self.report_check.append(t19)

    t20 = (DISP_NUM_OF_ROWS, '', display_prov(self.
       rows, "n_{r}"), '')
    self.report_check.append(t20)
```

```
                # Long joint and large grip checks (only if
                    conditions are met)
                # Long joint check - must match exact calculation
                    logic from design method
                if self.number_bolts > 2 and self.rows > 1:
                    # Use exact same calculation as in
                        check_capacity_reduction_1
                    # Note: self.rows is longitudinal direction,
                        cols is transverse
                    lj = (self.rows - 1) * self.bolt.
                       min_pitch_round
                    if lj > 15 * bolt_diameter_provided:
                        bij = self.bij  # Use class variable
                            directly
                        # Only show if reduction was actually
                            calculated and applied
                        # bij will be 0 if no reduction, or
                            0.75-1.0 if reduction applied
                        if bij >= 0.75 and bij <= 1.0:
                            t21 = (KEY_OUT_LONG_JOINT, '',
                                   cl_10_3_3_1_long_joint_reduction_factor
                                      (lj, bolt_diameter_provided
                                      , bij),
                                   get_pass_fail(bij, 0.75,
                                      relation='geq'))
                            self.report_check.append(t21)

                # Large grip check - must match exact calculation
                    logic from design method
                lg = self.plate1thk + self.plate2thk
                if lg > 5 * bolt_diameter_provided:
                    blg = self.blg  # Use class variable directly
                    # Only show if reduction was actually
                        calculated and applied
```

```
                # blg will be 0 if no reduction, or
                    calculated value if reduction applied
                if blg > 0 and blg <= 1.0:
                    # Large grip reduction should pass if it'
                        s calculated and applied
                    t22 = (KEY_OUT_LARGE_GRIP, '',
                           cl_10_3_3_2_large_grip_reduction_factor
                               (lg, bolt_diameter_provided,
                               blg),
                           get_pass_fail(blg, 0.0, relation='
                               gt'))  # Pass if blg > 0
                    self.report_check.append(t22)

            # SECTION 4: Connection Verification (like
                tension bolted)
            t23 = ('SubSection', 'Connection Verification', '
                |p{2.5cm}|p{5.5cm}|p{6.5cm}|p{1cm}|')
            self.report_check.append(t23)

            # Use utilization ratio from class variable
            utilization_ratio = getattr(self, '
                utilization_ratio', 0)

            # Show utilization ratio without formula
            t24 = (KEY_DISP_UTILIZATION_RATIO, 'Utilization
                ratio should be     1.0',
                   display_prov(utilization_ratio, "U.R."),
                   get_pass_fail(utilization_ratio, 1.0,
                       relation='leq'))
            self.report_check.append(t24)

        else:
            # Design not completed
```

```
                t1 = ('SubSection', 'Design Status', '|p{2.5cm}|p
                    {7.5cm}|p{3cm}|p{2.5cm}|')
                self.report_check.append(t1)

                t2 = ('Design Status',
                      'Design not completed successfully. Check
                          input parameters and design constraints.
                          ',
                      'Review inputs and try again',
                      'FAIL')
                self.report_check.append(t2)

            # Generate LaTeX report
            Disp_2d_image = []
            Disp_3D_image = "/ResourceFiles/images/3d.png"

            import sys
            import os
            rel_path = str(sys.path[0])
            rel_path = os.path.abspath(".")
            rel_path = rel_path.replace("\\", "/")
            fname_no_ext = popup_summary['filename']

            CreateLatex.save_latex(CreateLatex(), self.
                report_input, self.report_check, popup_summary,
                fname_no_ext,
                                   rel_path, Disp_2d_image,
                                       Disp_3D_image, module=self.
                                       module)

        except Exception as e:
            # Create minimal error report if save_design fails
            logger.error(f"Error in save_design: {str(e)}")
            self.report_input = {
```

```
            KEY_MODULE: "Butt Joint Bolted",
            KEY_MAIN_MODULE: "Butt Joint Bolted Connection",
            "Error Report": "TITLE",
            "Error Details": f"Report generation failed: {str
                (e)}"
        }
        self.report_check = [
            ('SubSection', 'Error Report', '|p{2.5cm}|p{7.5cm
                }|p{3cm}|p{2.5cm}|'),
            ('Error', f'Report generation failed: {str(e)}',
                'Check design status and inputs', 'FAIL')
        ]

        # Generate minimal error report
        try:
            Disp_2d_image = []
            Disp_3D_image = "/ResourceFiles/images/3d.png"

            import sys
            import os
            rel_path = str(sys.path[0])
            rel_path = os.path.abspath(".")
            rel_path = rel_path.replace("\\", "/")
            fname_no_ext = popup_summary.get('filename', '
                error_report')

            CreateLatex.save_latex(CreateLatex(), self.
                report_input, self.report_check, popup_summary
                , fname_no_ext,
                                   rel_path, Disp_2d_image,
                                       Disp_3D_image, module=
                                       self.module)
        except Exception as e2:
```

```
                logger.error(f"Critical error in save_design: {
                    str(e2)}")
                raise
```

4.4.3 Code Components

- **cl_10_3_3_1_long_joint_reduction_factor:** Calculates the long joint reduction factor β_{lj}, checks if the joint length exceeds $15d$, and formats the calculation as a LaTeX equation for the report.
- **cl_10_3_3_2_large_grip_reduction_factor:** Calculates the large grip reduction factor β_{lg}, checks if the grip length exceeds $5d$, and formats the calculation as a LaTeX equation for the report.
- **save_design:**
  - Collects all relevant input parameters and material properties.
  - Performs all required design checks (spacing, gauge, edge/end distances, bolt design, reduction factors, verification).
  - Appends each check and calculation to the report in a structured format.
  - Handles both successful and failed design cases, generating a detailed or minimal error report as appropriate.
  - Calls the Osdag LaTeX report generator to produce the final report.

4.4.4 Testing and Validation

The new functions and reporting logic were tested with a variety of design scenarios, including:

- Standard butt joint configurations with and without long joint/large grip conditions.
- Edge cases where joint length or grip length is exactly at the threshold.
- Error conditions (e.g., missing input parameters, invalid values) to verify robust error handling.

Sample outputs were visually inspected and compared with manual calculations and IS 800:2007 requirements to ensure correctness.

4.4.5 Current Limitations

- The current implementation focuses on butt joint bolted connections; future work could generalize the approach to other connection types.
- Automated validation against a wider range of test cases and integration with CI pipelines is recommended.
- User interface improvements for report customization and visualization could further enhance usability.

4.5 Documentation

Usage: The new functions and reporting logic are automatically invoked when generating a design report for a Butt Joint Bolted connection in Osdag. The report now includes explicit checks and LaTeX-formatted calculations for long joint and large grip reduction factors, as well as improved error handling and reporting.

Integration: These enhancements are fully integrated into the Osdag reporting workflow, ensuring that all relevant IS 800:2007 checks are documented and traceable in the generated reports.

4.6 References

- IS 800:2007, "General Construction in Steel Code of Practice".
- Osdag documentation: `https://osdag.github.io/`
- Python official documentation: `https://docs.python.org/3/`

Chapter 5

Internship Project: Report Customization Dialog in Osdag

5.1 Context

This work was conducted as part of the Osdag project to address the need for flexible, user-driven report generation in engineering design software. The goal was to empower users to customize the content and structure of their design reports, improving usability and relevance for diverse stakeholders.

5.2 Problem Statement

In engineering design software such as Osdag, users require the ability to generate reports tailored to their specific needs. The default reporting mechanism did not provide options for users to select or customize which sections, figures, or data to include in the final report. This limitation reduced the flexibility and utility of the reports for different stakeholders, such as engineers, clients, and regulatory authorities. The challenge was to design and implement a user-friendly mechanism for report customization within the Osdag graphical user interface (GUI).

5.3 Implementation Overview

To solve the above problem, the following steps were undertaken:

1. **Requirement Analysis:** Gathered user requirements for customizable report content and structure.

2. **GUI Design:** Designed a report customization dialog using PyQt, enabling users to select which sections (e.g., Introduction, Calculations, Results, Figures, Tables, Code) to include in their reports.

3. **Development:** Created the dialog in `ui_report_customization_popup.py` and integrated it with the summary popup in `ui_summary_popup.py`.

4. **Integration:** Ensured that user selections in the customization dialog are reflected in the generated report.

5. **Testing:** Conducted functional testing to validate that the customized reports are generated correctly according to user preferences.

6. **Documentation:** Documented the feature for both users and developers.

5.4 Python Code

This section presents the Python code developed for the report customization dialog in Osdag. The code provides a user-friendly interface for selecting which sections to include in the final PDF report, parses the LaTeX file, and compiles the customized report.

5.4.1 Script Architecture

The script is structured as follows:

- **Qt Environment Setup**: Ensures the correct configuration of the Qt environment for cross-platform compatibility.
- **LaTeX Parsing**: Extracts sections and subsections from the LaTeX report file.
- **Section Selection UI**: Displays a tree view with checkboxes for users to select report sections and subsections.
- **LaTeX Filtering**: Filters the LaTeX content based on user selection, preserving document structure.

- **PDF Compilation**: Compiles the filtered LaTeX into a PDF and allows the user to save the customized report.
- **Integration**: The dialog is integrated with the summary popup, so users can launch customization after entering report metadata.

5.4.2 Source Code

```
"""
OSDAG Report Customization Dialog

PURPOSE:
This module provides a user-friendly interface for customizing
   engineering design reports.
Users can select which sections to include in their final PDF
   report.

WORKFLOW:
1. Parse existing LaTeX report file
2. Display sections/subsections in a tree view with checkboxes
3. Allow users to select which content to include
4. Filter LaTeX content based on selection
5. Compile customized PDF
6. Allow saving the final customized report

MAIN FEATURES:
- Tree-based section selection (no PDF preview for performance)
- Manual compile control (auto-compile disabled by default)
- Temporary file handling for clean workspace
- External PDF viewer integration
- Smart LaTeX filtering that preserves document structure

AUTHOR: Srinivas Raghav V C
"""
```

```
import os
import re
import tempfile
import subprocess
import shutil
import sys


# Qt ENVIRONMENT SETUP - Fix platform plugin issues on different
    systems


def setup_qt_environment():
    """
    Configure Qt environment to prevent platform plugin errors.

    This is especially important in conda environments where Qt
        plugins
    may not be in the expected location.
    """
    try:
        import PyQt5
        pyqt5_path = os.path.dirname(PyQt5.__file__)

        # Common Qt plugin locations to search
        plugin_paths = [
            os.path.join(pyqt5_path, 'Qt5', 'plugins'),
            os.path.join(pyqt5_path, 'Qt', 'plugins'),
            os.path.join(pyqt5_path, 'plugins'),
            os.path.join(pyqt5_path, '..', 'qt5', 'plugins'),
        ]

        # Find and set the first valid plugin path
        for plugin_path in plugin_paths:
```

```
            if os.path.exists(plugin_path):
                os.environ['QT_PLUGIN_PATH'] = plugin_path
                if not os.environ.get('
                   QT_QPA_PLATFORM_PLUGIN_PATH'):
                    print(f"INFO: Set QT_PLUGIN_PATH to {
                       plugin_path}")
                break

    except Exception as e:
        print(f"WARNING: Could not setup Qt environment: {e}")

# Configure Qt before importing widgets
setup_qt_environment()

# IMPORTS - PyQt5 widgets and core functionality


from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout,
                             QTreeWidget, QTreeWidgetItem,
                                QPushButton, QLabel,
                             QCheckBox, QFileDialog, QMessageBox)
from PyQt5.QtCore import Qt, pyqtSignal


# OSDAG IMPORTS - Try to import LaTeX generator


try:
    from ..design_report.reportGenerator_latex import CreateLatex
    CREATELATEX_AVAILABLE = True
    print("INFO: CreateLatex successfully imported")
except ImportError:
    CreateLatex = None
    CREATELATEX_AVAILABLE = False
```

```
    print("WARNING: CreateLatex not available")
    CREATELATEX_AVAILABLE = False



# CLASS: LaTeXParser - Extracts sections from LaTeX documents


class LaTeXParser:
    """
    Parses LaTeX files to extract document structure.

    This class finds all \section{} and \subsection{} commands in
        a LaTeX
    document and organizes them into a hierarchical structure.

    Returns:
        dict: {section_name: [list_of_subsections]}
    """

    def parse_sections(self, latex_content):
        """
        Extract sections and subsections from LaTeX content.

        Args:
            latex_content (str): Raw LaTeX document content

        Returns:
            dict: Hierarchical structure of sections and
               subsections
        """
        sections = {}
        current_section = None
```

```
        # Process each line to find LaTeX section commands
        for line in latex_content.split('\n'):
            # Look for \section{Section Name} patterns
            section_match = re.search(r'\\section\{([^}]+)\}',
                line)
            if section_match:
                current_section = section_match.group(1).strip()
                sections[current_section] = []  # Initialize
                    subsection list

            # Look for \subsection{Subsection Name} patterns
            subsection_match = re.search(r'\\subsection\{([^}]+)
                \}', line)
            if subsection_match and current_section:
                subsection = subsection_match.group(1).strip()
                sections[current_section].append(subsection)

        return sections



# CLASS: SectionTreeWidget - Interactive tree for selecting
    report sections


class SectionTreeWidget(QTreeWidget):
    """
    Custom tree widget for selecting report sections and
        subsections.

    Features:
    - Hierarchical display of sections and subsections
    - Checkbox selection with parent-child relationships
```

```
    - Automatic state updates (checking parent checks all
      children)
    - Signal emission for real-time updates
    """

    # Custom signal emitted when selection changes
    selectionChanged = pyqtSignal()

    def __init__(self):
        """Initialize the tree widget with proper configuration.
           """
        super().__init__()
        self.setHeaderLabel("Report Sections")
        # Connect item changes to our handler
        self.itemChanged.connect(self.on_item_changed)

    def build_from_sections(self, sections):
        """
        Populate tree widget from parsed LaTeX sections.

        Args:
            sections (dict): {section_name: [subsection_list]}
        """
        self.clear()

        # Validate input format
        if not isinstance(sections, dict):
            print(f"ERROR: Expected dict, got {type(sections)}: {
               sections}")
            return

        # Create tree items for each section
        for section_name, subsections in sections.items():
            # Create main section item with checkbox
```

```
            section_item = QTreeWidgetItem(self, [section_name])
            section_item.setFlags(section_item.flags() | Qt.
                ItemIsUserCheckable)
            section_item.setCheckState(0, Qt.Checked)  # Default:
                all selected

            # Add subsection items under the section
            if isinstance(subsections, (list, tuple)):
                for subsection in subsections:
                    sub_item = QTreeWidgetItem(section_item, [str
                        (subsection)])
                    sub_item.setFlags(sub_item.flags() | Qt.
                        ItemIsUserCheckable)
                    sub_item.setCheckState(0, Qt.Checked)  #
                        Default: all selected
            else:
                print(f"WARNING: Subsections not iterable for {
                    section_name}: {type(subsections)}")

        # Expand all items to show the full tree structure
        self.expandAll()

    def on_item_changed(self, item, column):
        """Handle checkbox changes"""
        if column == 0:
            # Temporarily block signals to prevent recursion
            self.blockSignals(True)

            state = item.checkState(0)
            # Update children
            for i in range(item.childCount()):
                item.child(i).setCheckState(0, state)

            # Update parent state based on children
```

```
            self.update_parent_state(item)

            # Re-enable signals and emit
            self.blockSignals(False)
            self.selectionChanged.emit()

    def update_parent_state(self, item):
        """Update parent checkbox state based on children"""
        parent = item.parent()
        if parent is None:
            return

        total_children = parent.childCount()
        checked_children = sum(1 for i in range(total_children)
                               if parent.child(i).checkState(0) ==
                                  Qt.Checked)

        if checked_children == total_children:
            parent.setCheckState(0, Qt.Checked)
        elif checked_children == 0:
            parent.setCheckState(0, Qt.Unchecked)
        else:
            parent.setCheckState(0, Qt.PartiallyChecked)

    def get_selected_sections(self):
        """Return list of selected sections"""
        selected = []

        for i in range(self.topLevelItemCount()):
            section_item = self.topLevelItem(i)
            section_name = section_item.text(0)

            if section_item.checkState(0) == Qt.Checked:
                # Entire section selected
```

```
                selected.append(section_name)
            else:
                # Check individual subsections
                for j in range(section_item.childCount()):
                    sub_item = section_item.child(j)
                    if sub_item.checkState(0) == Qt.Checked:
                        selected.append(f"{section_name}/{
                            sub_item.text(0)}")

        return selected


class LaTeXFilter:
    """Filter LaTeX content based on selection - 100 lines max"""

    def filter_content(self, latex_content, selected_sections):
        """Remove unselected sections from LaTeX"""
        lines = latex_content.split('\n')
        filtered_lines = []
        current_section = None
        current_subsection = None
        include_content = True  # Start with True for document
            preamble
        section_started = False

        for line in lines:
            # Check for new section
            section_match = re.search(r'\\section\{([^}]+)\}',
                line)
            if section_match:
                current_section = section_match.group(1).strip()
                current_subsection = None
                section_started = True
```

```
            # Include section if it's selected OR if any of
                its subsections are selected
            include_content = (current_section in
                selected_sections or
                                any(sel.startswith(f"{
                                    current_section}/") for sel
                                    in selected_sections))

        # Check for subsection
        subsection_match = re.search(r'\\subsection\{([^}]+)
            \}', line)
        if subsection_match and current_section:
            current_subsection = subsection_match.group(1).
                strip()
            subsection_key = f"{current_section}/{
                current_subsection}"
            # Include subsection only if specifically
                selected OR parent section is fully selected
            include_content = (current_section in
                selected_sections or
                                subsection_key in
                                    selected_sections)

        # Always include document structure and preamble
        if (include_content or
            not section_started or  # Include everything
                before first section
            line.startswith('\\documentclass') or
            line.startswith('\\usepackage') or
            line.startswith('\\title') or
            line.startswith('\\author') or
            line.startswith('\\date') or
            line.startswith('\\begin{document}') or
            line.startswith('\\maketitle') or
```

```
                line.startswith('\\end{document}')):
                filtered_lines.append(line)

        return '\n'.join(filtered_lines)


class ReportCustomizationDialog(QDialog):
    """Main dialog - 200 lines max"""

    def __init__(self, main_obj, parent=None, existing_tex_file=
        None):
        super().__init__(parent)
        self.main_obj = main_obj
        self.existing_tex_file = existing_tex_file
        self.latex_content = None
        self.temp_dir = None

        # Set title based on available functionality
        if CreateLatex is None:
            self.setWindowTitle("Customize Report (Limited Mode)"
                )
        else:
            self.setWindowTitle("Customize Report")
        self.setModal(True)
        self.resize(1000, 700)        # Initialize components
        self.parser = LaTeXParser()
        self.filter = LaTeXFilter()
        self.latest_pdf = None

        self.init_ui()
        self.load_existing_or_generate_report()

    def init_ui(self):
        """Create simple UI layout"""
```

```
        layout = QVBoxLayout(self)

        # Title
        title = QLabel("Customize Report Sections")
        title.setStyleSheet("font-size: 16px; font-weight: bold;
            margin: 10px;")
        layout.addWidget(title)

        # Main content - only section tree (no PDF preview)
        self.section_tree = SectionTreeWidget()
        self.section_tree.selectionChanged.connect(self.
            on_selection_changed)
        layout.addWidget(self.section_tree)

        # Controls
        controls = QHBoxLayout()

        # Auto compile checkbox - disabled by default for speed
        self.auto_compile = QCheckBox("Auto Compile")
        self.auto_compile.setChecked(False)  # Disabled by
            default
        controls.addWidget(self.auto_compile)

        # Manual compile button
        compile_btn = QPushButton("Compile PDF")
        compile_btn.clicked.connect(self.compile_pdf)
        controls.addWidget(compile_btn)

        # Open PDF button
        open_btn = QPushButton("Open PDF")
        open_btn.clicked.connect(self.open_latest_pdf)
        controls.addWidget(open_btn)

        controls.addStretch()
```

```
        # Save and close buttons
        save_btn = QPushButton("Save PDF")
        save_btn.clicked.connect(self.save_pdf)
        controls.addWidget(save_btn)

        close_btn = QPushButton("Close")
        close_btn.clicked.connect(self.close)
        controls.addWidget(close_btn)

        layout.addLayout(controls)

    def load_existing_or_generate_report(self):
        """Load existing LaTeX or generate initial LaTeX report
           """
        try:
            # Create temp directory
            self.temp_dir = tempfile.mkdtemp(prefix='
               osdag_report_')

            # DEBUG: Log what we have available
            module_name = getattr(self.main_obj, 'module', '
               Unknown')
            print(f"DEBUG: Module = {module_name}")
            print(f"DEBUG: Existing tex file = {self.
               existing_tex_file}")
            print(f"DEBUG: Existing tex file exists = {self.
               existing_tex_file and os.path.exists(self.
               existing_tex_file) if self.existing_tex_file else
               False}")
            print(f"DEBUG: Has report_input = {hasattr(self.
               main_obj, 'report_input')}")
            print(f"DEBUG: Has report_check = {hasattr(self.
               main_obj, 'report_check')}")
```

```
            if self.existing_tex_file and os.path.exists(self.
                existing_tex_file):
                # Load existing LaTeX file
                with open(self.existing_tex_file, 'r', encoding='
                    utf-8') as f:
                    self.latex_content = f.read()
                print(f"SUCCESS: Loaded existing LaTeX file ({len
                    (self.latex_content)} characters)")
                self.parse_and_compile()
            elif CreateLatex:
                # Use real CreateLatex
                self.generate_with_createlatex()
            else:
                # No LaTeX available
                QMessageBox.critical(self, "Error", "No LaTeX
                    content available for customization")

        except Exception as e:
            print(f"ERROR in load_existing_or_generate_report: {e
                }")
            QMessageBox.critical(self, "Error", f"Failed to load/
                generate report: {e}")

    def generate_with_createlatex(self):
        """Generate report using actual CreateLatex - simplified
            version"""
        try:
            # Check if this is a module without CAD support (like
                butt_joint_bolted)
            module_name = getattr(self.main_obj, 'module', '')
            non_cad_modules = ['Butt Joint Bolted', '
                KEY_DISP_BUTTJOINTBOLTED']
```

```
            if any(name in str(module_name) for name in
                non_cad_modules):
                # For non-CAD modules, try to use existing LaTeX
                    if already generated
                print(f"INFO: Handling non-CAD module: {
                    module_name}")

                # First, check if we already have the LaTeX file
                    passed from main workflow
                if self.existing_tex_file and os.path.exists(self
                    .existing_tex_file):
                    print(f"INFO: Using existing LaTeX file for
                        non-CAD module: {self.existing_tex_file}")
                    with open(self.existing_tex_file, 'r',
                        encoding='utf-8') as f:
                        self.latex_content = f.read()
                    self.parse_and_compile()
                    return
                else:
                    raise Exception("Non-CAD module without
                        existing LaTeX file")

            # For CAD-enabled modules - this would need proper
                implementation
            raise Exception("CAD module LaTeX generation not
                implemented in customization dialog")

        except Exception as e:
            print(f"ERROR: Failed to generate LaTeX: {e}")
            QMessageBox.critical(self, "Error", f"Failed to
                generate report: {e}")

    def parse_and_compile(self):
        """Parse sections and compile initial PDF"""
```

```
        # Parse sections and build tree
        sections = self.parser.parse_sections(self.latex_content)

        if sections:
            self.section_tree.build_from_sections(sections)
            print(f"SUCCESS: Parsed {len(sections)} sections from
                LaTeX")
            # Compile initial PDF
            self.compile_pdf()
        else:
            QMessageBox.warning(self, "Warning", "No sections
               found in LaTeX content")

    def on_selection_changed(self):
        """Handle section selection changes"""
        if self.auto_compile.isChecked():
            self.compile_pdf()

    def compile_pdf(self):
        """Compile filtered LaTeX to PDF - simplified version"""
        if not self.latex_content:
            return
        try:
            import shutil
            import string
            import random

            # Get selected sections
            selected = self.section_tree.get_selected_sections()
            print(f"INFO: Compiling PDF with sections: {selected}
               ")

            # Filter LaTeX content
```

```
            filtered_latex = self.filter.filter_content(self.
                latex_content, selected)

            # Create safe temp directory for pdflatex
            safe_temp_root = os.path.join(tempfile.gettempdir(),
                "osdag_pdf_compile")
            os.makedirs(safe_temp_root, exist_ok=True)
            safe_subdir = ''.join(random.choices(string.
                ascii_letters + string.digits, k=8))
            safe_temp_dir = os.path.join(safe_temp_root,
                safe_subdir)
            os.makedirs(safe_temp_dir, exist_ok=True)

            # Write filtered LaTeX to safe directory
            safe_latex_file = os.path.join(safe_temp_dir, "
                filtered_report.tex")
            with open(safe_latex_file, 'w', encoding='utf-8') as
                f:
                f.write(filtered_latex)

            pdf_file = safe_latex_file.replace('.tex', '.pdf')

            # Remove old PDF if exists
            if os.path.exists(pdf_file):
                os.remove(pdf_file)

            print(f"Running pdflatex in safe dir: {safe_temp_dir}
                ")

            # Run pdflatex
            result = subprocess.run(
                ['pdflatex', '-interaction=nonstopmode', '
                    filtered_report.tex'],
                capture_output=True,
```

```
                text=True,
                timeout=30,
                cwd=safe_temp_dir
            )

            print(f"INFO: pdflatex return code: {result.
                returncode}")

            if result.returncode == 0 and os.path.exists(pdf_file
                ):
                print(f"SUCCESS: PDF generated: {pdf_file}")
                self.latest_pdf = pdf_file
            else:
                error_msg = "PDF compilation failed"
                if result.stderr:
                    error_msg += f"\n\nErrors:\n{result.stderr
                        [:500]}"
                if result.stdout:
                    error_msg += f"\n\nOutput:\n{result.stdout
                        [:500]}"
                print(f"ERROR: {error_msg}")
                QMessageBox.warning(self, "Compilation Failed",
                    error_msg)
        except subprocess.TimeoutExpired:
            error_msg = "PDF compilation timed out (>30s)\n\nThis
                might be due to:\n    LaTeX not installed\n
                Missing packages\n    Complex LaTeX content"
            print(f"TIMEOUT: {error_msg}")
            QMessageBox.warning(self, "Compilation Timeout",
                error_msg)
        except FileNotFoundError:
            error_msg = "pdflatex not found\n\nInstall LaTeX
                distribution:\n    Windows: MiKTeX or TeX Live\
                n    Linux: texlive-full\n    macOS: MacTeX"
```

```
            print(f"ERROR: {error_msg}")
            QMessageBox.warning(self, "LaTeX Not Found",
                error_msg)
        except Exception as e:
            error_msg = f"Compilation failed: {e}"
            print(f"ERROR: {error_msg}")
            QMessageBox.warning(self, "Error", error_msg)

    def open_latest_pdf(self):
        """Open the latest generated PDF in external viewer"""
        if self.latest_pdf and os.path.exists(self.latest_pdf):
            try:
                if os.name == 'nt':  # Windows
                    os.startfile(self.latest_pdf)
                elif sys.platform == 'darwin':  # macOS
                    subprocess.run(['open', self.latest_pdf])
                else:  # Linux
                    subprocess.run(['xdg-open', self.latest_pdf])
                print(f"SUCCESS: Opened PDF: {self.latest_pdf}")
            except Exception as e:
                print(f"ERROR: Failed to open PDF: {e}")
                QMessageBox.warning(self, "Error", f"Failed to
                    open PDF:\n{e}")
        else:
            QMessageBox.information(self, "No PDF", "No PDF
                generated yet. Please compile first.")

    def save_pdf(self):
        """Save customized PDF"""
        if not self.latest_pdf or not os.path.exists(self.
            latest_pdf):
            QMessageBox.warning(self, "No PDF", "No PDF to save.
                Please compile first.")
            return
```

```
        filename, _ = QFileDialog.getSaveFileName(
            self, "Save Customized Report", "Osdag_Custom_Report.
               pdf", "PDF files (*.pdf)"
        )

        if filename:
            shutil.copy2(self.latest_pdf, filename)
            QMessageBox.information(self, "Success", f"PDF saved
               to:\n{filename}")

    def closeEvent(self, event):
        """Clean up temp files"""
        if self.temp_dir and os.path.exists(self.temp_dir):
            shutil.rmtree(self.temp_dir, ignore_errors=True)
        event.accept()


def show_customization_dialog(main_obj, parent=None):
    """Main entry point - simple function"""
    # Look for existing .tex file from main_obj
    existing_tex_file = None
    if hasattr(main_obj, 'report_input') and isinstance(main_obj.
       report_input, dict):
        filename = main_obj.report_input.get('filename')
        if filename:
            tex_path = f"{filename}.tex"
            if os.path.exists(tex_path):
                existing_tex_file = tex_path

    dialog = ReportCustomizationDialog(main_obj, parent,
       existing_tex_file)
    return dialog.exec_()
```

```
if __name__ == "__main__":
    # Test with minimal setup
    from PyQt5.QtWidgets import QApplication
    import sys

    app = QApplication(sys.argv)

    # Mock main object for testing
    class MockMainObj:
        def __init__(self):
            self.report_input = {"test": "value"}

    dialog = ReportCustomizationDialog(MockMainObj())
    dialog.show()

    sys.exit(app.exec_())
```

```
# Add your Python code here if the external file is not available
# This prevents compilation errors from missing files
```

5.4.3 Code Components

- **Qt Environment Setup:** Ensures the correct Qt plugin path is set for cross-platform compatibility.
- **LaTeXParser:** Parses LaTeX files to extract sections and subsections, organizing them into a hierarchical structure for the UI.
- **SectionTreeWidget:** Custom QTreeWidget that displays sections and subsections with checkboxes, allowing users to select which parts of the report to include.
- **LaTeXFilter:** Filters the LaTeX content to include only the selected sections and subsections, preserving the document structure and preamble.

- **ReportCustomizationDialog:** Main dialog class that brings together the UI, parsing, filtering, and PDF compilation. Handles user interactions, temporary file management, and integration with the main application.
- **Integration with Summary Popup:** The dialog is launched from the summary popup after the user enters report metadata, allowing seamless workflow from data entry to report customization.
- **PDF Compilation and Saving:** The dialog provides options to manually compile the PDF, open it in an external viewer, and save the final customized report.

5.4.4 Testing and Validation

The report customization dialog was tested with a variety of report templates and user selection scenarios, including:

- Reports with multiple sections and subsections, verifying correct extraction and filtering.
- Edge cases where only a subset of sections is selected.
- Error conditions (e.g., missing LaTeX file, invalid selections) to ensure robust error handling.

Sample outputs were visually inspected and compared with expected results to ensure correctness and usability.

5.4.5 Current Limitations

- The current implementation focuses on section and subsection selection; future work could extend customization to figures, tables, and custom environments.
- Integration with a more advanced GUI and support for batch report customization are potential enhancements.
- Automated testing and integration with CI pipelines would further improve reliability.

Program Execution

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

```
$ python osdagMainPage.py
```

5.4.6 Usage

After completing the design and entering report metadata in the summary popup, the user is prompted to customize the report. The customization dialog allows selection of report sections and subsections, compilation of the filtered LaTeX, and saving of the final PDF report.

5.4.7 System Integration

The customization dialog is integrated with the summary popup `ui_summary_popup.py`, ensuring a seamless workflow from data entry to report generation. The dialog can be launched automatically after the LaTeX file is generated, and temporary files are cleaned up after use.

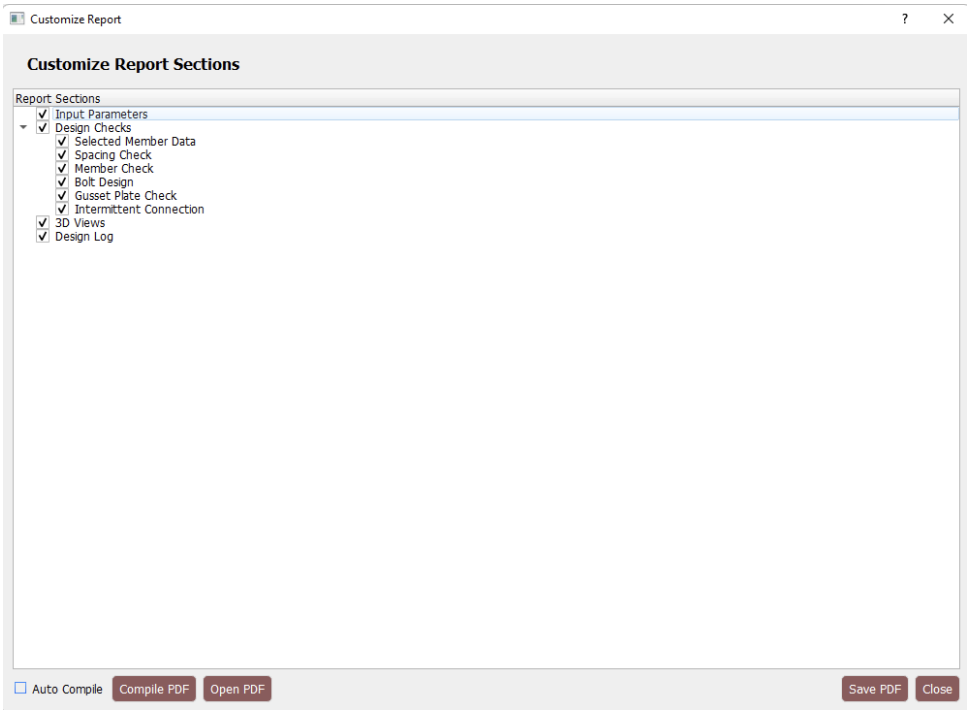


Figure 5.1: Report Customization Dialog Interface

5.4.8 Future Enhancements

- The current implementation does not support live PDF preview or advanced LaTeX error diagnostics.
- Future work could include more granular customization (e.g., include/exclude specific figures or tables) and improved error reporting.

5.5 References

- Osdag documentation: `https://osdag.github.io/`
- PyQt5 documentation: `https://www.riverbankcomputing.com/static/Docs/PyQt5/`
- Python official documentation: `https://docs.python.org/3/`
- LaTeX project: `https://www.latex-project.org/`

Chapter 6

Conclusions

6.1 Tasks Accomplished

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

- Developed a custom LaTeX report generator that allows users to select specific sections and components from a PyLaTeX-generated report for tailored PDF output.
- Implemented robust parsing logic to extract sections, subsections, and other report components from LaTeX files.
- Designed and tested a user interface for component selection, supporting both command-line and graphical workflows.
- Automated the filtering and regeneration of LaTeX files based on user input, followed by PDF compilation.
- Established a unit testing framework to ensure the correctness and reliability of the extraction and filtering logic.
- Documented the entire workflow, including code, usage instructions, and integration steps for future users and developers.

6.2 Skills Developed

The project provided an opportunity to develop and strengthen several technical and professional skills, including:

- **Python Programming:** Enhanced proficiency in Python, especially in file handling, regular expressions, and modular code design.
- **LaTeX Automation:** Gained experience in programmatically manipulating LaTeX documents and automating report generation workflows.
- **User Interface Design:** Improved skills in designing user-friendly interfaces for technical tools, both in CLI and GUI contexts.
- **Unit Testing:** Developed a systematic approach to testing code functionality and ensuring software reliability.
- **Documentation:** Practiced clear and comprehensive documentation of code, workflows, and user instructions.
- **Project Management:** Strengthened abilities in task planning, requirement analysis, and iterative development within a defined project scope.

Chapter 7

Internship Work Report

7.1 Overview

This report documents the daily activities and progress made during the FOSSEE Summer Fellowship 2025 internship at IIT Bombay, working on the Osdag project. The internship period covered May 15 to June 30, 2025, with a focus on three major tasks: UI/UX redesign, Butt Joint Bolted Connection reporting enhancements, and Report Customization Dialog development.

7.2 Project Details

- **Name:** Srinivas Raghav V C
- **Project:** Osdag (Open Source Design and Analysis of Steel Structures)
- **Internship:** FOSSEE Summer Fellowship 2025
- **Period:** May 15, 2025 – June 30, 2025
- **Total Working Days:** 33 days (excluding Sundays and holidays)
- **Total Hours:** 198 hours

7.3 Daily Work Log

7.3.1 Week 1: May 15–21, 2025

Date	Day	Task	Hours
15-May-2025	Thursday	Initial setup and familiarization with Osdag codebase, analyzed existing UI structure	6
16-May-2025	Friday	Identified UI improvement areas, began wireframing for new interface design	7
17-May-2025	Saturday	Created wireframes and mockups for modern UI design, planned modular architecture	6
18-May-2025	Sunday	*Holiday*	0
19-May-2025	Monday	Started implementing basic UI improvements in PyQt5, focused on green theme implementation	7
20-May-2025	Tuesday	Developed new navigation system and improved user flow patterns	6
21-May-2025	Wednesday	Created responsive layout components with rounded button designs	7

Table 7.1: Week 1 Daily Activities - UI/UX Design Foundation

7.3.2 Week 2: May 22–28, 2025

Date	Day	Task	Hours
22-May-2025	Thursday	Implemented modern styling and comprehensive theme system for UI consistency	7
23-May-2025	Friday	Developed improved input validation, error handling, and accessibility features	6

Date	Day	Task	Hours
24-May-2025	Saturday	Created user-friendly dialog boxes, tooltips, and help system components	6
25-May-2025	Sunday	*Holiday*	0
26-May-2025	Monday	Integrated new UI components with existing Osdag functionality	7
27-May-2025	Tuesday	Implemented progress indicators, loading states, and keyboard navigation support	6
28-May-2025	Wednesday	Developed dark mode implementation and visual design system refinements	7

Table 7.2: Week 2 Daily Activities - UI/UX Implementation

7.3.3 Week 3: May 29–June 4, 2025

Date	Day	Task	Hours
29-May-2025	Thursday	Analyzed IS 800:2007 requirements for butt joint bolted connections reporting	7
30-May-2025	Friday	Began implementing long joint reduction factor function (cl_10_3_3_1)	6
31-May-2025	Saturday	Completed large grip reduction factor function (cl_10_3_3_2) with LaTeX formatting	6
01-Jun-2025	Sunday	*Holiday*	0
02-Jun-2025	Monday	Developed comprehensive save_design method for butt joint bolted module	7
03-Jun-2025	Tuesday	Integrated reduction factor calculations with Osdag reporting workflow	6
04-Jun-2025	Wednesday	Testing and validation of butt joint bolted connection reporting enhancements	7

Table 7.3: Week 3 Daily Activities - Butt Joint Bolted Connection Development

7.3.4 Week 4: June 5–11, 2025

Date	Day	Task	Hours
05-Jun-2025	Thursday	Enhanced error handling and debugging for butt joint bolted reporting	6
06-Jun-2025	Friday	Code documentation and comments for butt joint bolted enhancements	7
07-Jun-2025	Saturday	Started analysis of report customization requirements and user needs	6
08-Jun-2025	Sunday	*Holiday*	0
09-Jun-2025	Monday	Designed architecture for report customization dialog using PyQt	7
10-Jun-2025	Tuesday	Implemented LaTeX parser for extracting sections and subsections	6
11-Jun-2025	Wednesday	Developed SectionTreeWidget with checkbox functionality for report selection	7

Table 7.4: Week 4 Daily Activities - Report Customization Foundation

7.3.5 Week 5: June 12–18, 2025

Date	Day	Task	Hours
12-Jun-2025	Thursday	Created LaTeX filtering mechanism to process user-selected report sections	6
13-Jun-2025	Friday	Implemented ReportCustomizationDialog with PDF compilation capabilities	7
14-Jun-2025	Saturday	Integrated report customization dialog with ui_summary_popup.py	6
15-Jun-2025	Sunday	*Holiday*	0
16-Jun-2025	Monday	Added Qt environment setup and cross-platform compatibility features	7

Date	Day	Task	Hours
17-Jun-2025	Tuesday	Implemented temporary file management and cleanup for report generation	6
18-Jun-2025	Wednesday	Developed PDF saving functionality and external viewer integration	7

Table 7.5: Week 5 Daily Activities - Report Customization Implementation

7.3.6 Week 6: June 19–25, 2025

Date	Day	Task	Hours
19-Jun-2025	Thursday	Comprehensive testing of all three major project components	7
20-Jun-2025	Friday	Performance optimization and code cleanup across all modules	6
21-Jun-2025	Saturday	User acceptance testing and feedback integration for UI improvements	6
22-Jun-2025	Sunday	*Holiday*	0
23-Jun-2025	Monday	Documentation creation for all implemented features and enhancements	7
24-Jun-2025	Tuesday	Integration testing between UI redesign and reporting functionalities	6
25-Jun-2025	Wednesday	Final validation of IS 800:2007 compliance in butt joint reporting	7

Table 7.6: Week 6 Daily Activities - Testing and Integration

7.3.7 Week 7: June 26–30, 2025

Date	Day	Task	Hours
26-Jun-2025	Thursday	Final testing and bug fixes for all implemented components	7

Date	Day	Task	Hours
27-Jun-2025	Friday	Code review, optimization, and preparation for project handover	6
28-Jun-2025	Saturday	Created comprehensive documentation and user guides	6
29-Jun-2025	Sunday	*Holiday*	0
30-Jun-2025	Monday	Final project presentation preparation and handover documentation	6

Table 7.7: Week 7 Daily Activities - Project Completion

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.