



FOSSEE Spring 2026 Semester Long Internship Report

On

UI Development of tabs in additional inputs and
Generate Results Tab for Osdag

Submitted by

Jatin Nainwani

3rd Year B.Tech Student, Department of Engineering

VIT Bhopal University

Bhopal

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Nidhi Khare

Aditya Donde

Ajmal Babu M S

September 17, 2026

Acknowledgments

- I would like to express my sincere gratitude to everyone who supported and guided me throughout my internship and contributed to making this project a valuable learning experience.
- Special thanks to my mentors, **Aditya Donde** and **Nidhi Khare**, for their continuous guidance, valuable feedback, encouragement, and support throughout the internship. Their mentorship played an important role in helping me understand the project and overcome the challenges I encountered.
- I would also like to thank the fellow interns, **Garvit** and **Mohd. Suhail**, for their guidance, assistance, and willingness to share their knowledge and experience during my internship.
- I would like to express my gratitude to the project staff at the Osdag team, particularly **Ajmal Babu M. S.**, **Ajinkya Dahale**, and **Parth Karia**, for their support and assistance throughout the project.
- I sincerely thank the Osdag Principal Investigator (PI), **Prof. Siddhartha Ghosh**, Department of Civil Engineering, IIT Bombay, for providing the opportunity to work on the project and for his support towards the Osdag initiative.
- I would also like to acknowledge **Prof. Kannan M. Moudgalya**, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay, for his contribution and support to the FOSSEE project.
- I extend my sincere thanks to the FOSSEE Managers, **Usha Viswanathan** and **Vineeta Parmar**, and their entire team for their support and coordination throughout the internship.

- I gratefully acknowledge the support of the **National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India**, for their role in facilitating and supporting this project.
- I would also like to thank my colleagues and everyone who worked with me during my internship for their cooperation, support, and encouragement.
- Finally, I would like to express my gratitude to my college, department, faculty members, and everyone who supported me throughout my studies and helped me reach this stage.

Contents

1	Introduction	5
1.1	National Mission in Education through ICT	5
1.1.1	ICT Initiatives of MoE	6
1.2	FOSSEE Project	7
1.2.1	Projects and Activities	7
1.2.2	Fellowships	7
1.3	Osdag Software	8
1.3.1	Osdag GUI	9
1.3.2	Features	9
2	Screening Task	10
2.1	Problem Statement	10
2.2	Tasks Done	10
3	Additional Inputs Dialog: Schema-Driven Tabs	12
3.1	Task 1: Problem Statement	12
3.2	Task 1: Tasks Done	12
3.3	Task 1: Python Code	13
3.3.1	Description of the Script	13
3.3.2	Python Code	14
3.3.3	Explanation of the Code	15
4	Generate Results Dialog	16
4.1	Task 2: Problem Statement	16
4.2	Task 2: Tasks Done	17
4.2.1	Table hierarchy	17
4.2.2	Dialog design	17
4.2.3	Custom widgets	18
4.2.4	Populating the tables with user values	18
4.2.5	Excel export	18
4.2.6	Integration with the Output dock	19

4.3	Task 2: Documentation	19
4.3.1	Files	19
4.3.2	User workflow	19
4.3.3	Adding a new result table	19
5	Conclusions	21
5.1	Tasks Accomplished	21
5.2	Skills Developed	22
A	Appendix	24
A.1	Work Reports	24
	Bibliography	27

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	SWAYAMPBABHA	Access 24x7 TV programs	Enable SWAYAMPBABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

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

1.2 FOSSEE Project

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

1.2.1 Projects and Activities

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

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

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

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

For more details, visit the official FOSSEE website.

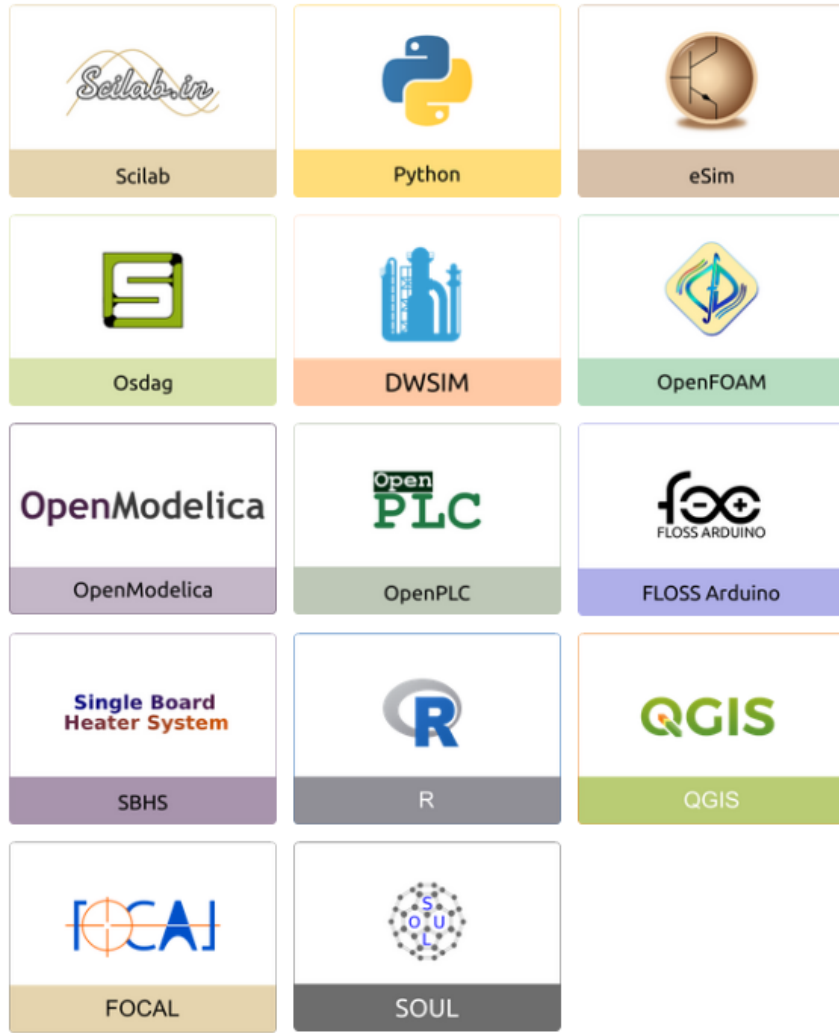


Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

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

1.3.1 Osdag GUI

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

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

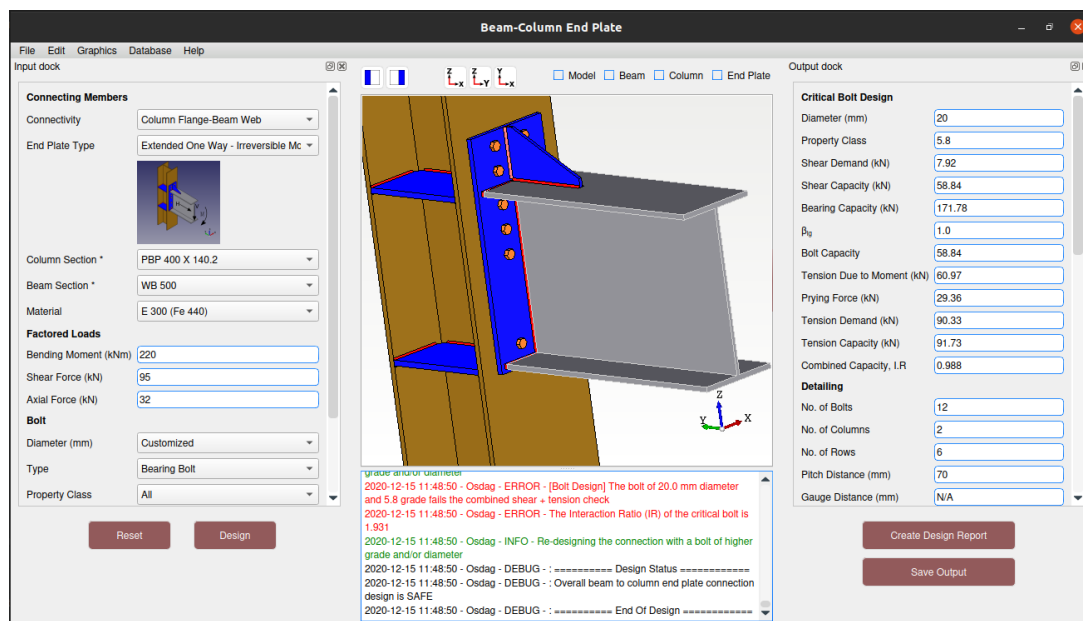


Figure 1.2: Osdag GUI

1.3.2 Features

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

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

The screening task involved setting up and developing a Dockerized full-stack application for managing and displaying location data consisting of states and their corresponding districts. The objective was to establish a working backend API, database, and frontend interface that could retrieve and display the available states and districts.

The application was designed using a Django backend with PostgreSQL as the database and a Node.js-based frontend. Docker and Docker Compose were used to containerize the different components of the application and ensure that the backend, frontend, and database could be configured and run together in a consistent environment.

2.2 Tasks Done

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

- **Project Setup:** Set up the screening-task project and configured the required development environment for the backend and frontend components.
- **Docker Configuration:** Configured Docker and Docker Compose to run the application using separate containers for the PostgreSQL database, Django backend, and frontend.
- **Database Setup:** Configured PostgreSQL as the database and established the required database configuration for the Django application.

- **Django Backend Development:** Set up the Django backend and created the required application structure for handling location-related data.
- **Location Models:** Implemented the required Django models for storing states and their corresponding districts.
- **Database Migrations:** Created and applied Django migrations to create the required database structure for the location data.
- **API Development:** Implemented the backend API for retrieving state and district information, including the states endpoint used by the frontend.
- **Frontend Integration:** Worked on the frontend implementation for retrieving the state data from the backend API and displaying it through the user interface.
- **Data Seeding:** Prepared the application to populate the database with the required state and district records so that the location information could be retrieved through the API.
- **Application Testing:** Verified the Docker containers, Django configuration, database connectivity, and backend application using Django system checks and API requests.
- **Full-Stack Integration:** Connected the frontend with the Django backend API and verified the communication between the application components.

Chapter 3

Additional Inputs Dialog: Schema-Driven Tabs

3.1 Task 1: Problem Statement

The Additional Inputs dialog of OsdagBridge (plate-girder module) collects secondary design data. Before this task it was one hand-written PySide6 file: labels, widgets, defaults and ranges were scattered through the GUI code, each field raised its own error pop-up, and the Defaults/Save buttons were unreliable. The task was to split the Support Conditions, Analysis/Design Options and Design Options (Cont.) tabs into separate files, generate their widgets from a single schema, validate each tab with one combined pop-up, and add a live CAD preview to the support tab.

3.2 Task 1: Tasks Done

Field definitions are plain Python dictionaries in `ui_fields_additional_input.py`. The parent dialog (`AdditionalInputs`) converts them into Qt widgets through `_create_schema_widget()` and `_build_sections_from_schema()`, and binds each widget as an attribute of the dialog. Each tab only lays out its cards and implements three hooks: `validate_tab()`, `save_values()` and `reset_defaults()`. The work flow is:

1. Schema dictionary (label, type, default, range, bind name)
2. Widget factory in the dialog creates the Qt widget and validator

3. Tab class places widgets in cards and adds tab-specific logic
4. On Save, the dialog validates every tab and merges their values

Tab	Field	Default	Range / Choices
Support Conditions	Left / Right Support Bearing Length (mm)	Pinned / Roller 400.00	Fixed, Pinned, Roller 0 – 600
Design Options	Reinforcement Size Reinforcement Material Shear stud f_y , f_u (MPa) Stud Diameter (mm) Stud Spacing (mm)	8–40 mm Fe 415 385, 495 20 schema	standard bars via Set Bounds Fe 415 / 500 / 550 350 – 600 12, 16, 20, 22, 25 $\geq 4 \times$ diameter
Design Options (Cont.)	Partial factors γ Load Cycles Deflection Limit L / Limit-state checks	IRC values 2000000 600 all on	1.0 – 2.0 integer range 300 – 800 checkbox groups

Table 3.1: Inputs defined in the three schemas.

Figure 3.1: Support Conditions tab with live CAD previews.

Figure 3.2: Analysis/Design Options and Design Options (Cont.) tabs.

3.3 Task 1: Python Code

Short excerpts of the four key pieces of the implementation are shown below.

3.3.1 Description of the Script

- **Schema:** each field is a dictionary with a label, type, default, optional validator and a bind name.
- **Widget factory:** converts a field dictionary into a Qt widget with its validator and exposes it on the dialog.
- **Tabs:** build one card per schema section; the support tab also connects the bearing-length field to a CAD preview.
- **Save:** the dialog validates every tab, shows all errors in one pop-up, then merges each tab’s saved values.

3.3.2 Python Code

Listing 3.1: One schema field (Shear Studs card)

```
1 {
2     "id": "shear_stud_yield_strength",
3     "label": "Yield Strength (MPa)",
4     "type": "line",
5     "default": "385.00",
6     "validator": {"type": "double_range",
7                   "bottom": 350, "top": 600, "decimals": 2},
8     "bind": "shear_stud_yield_strength_input",
9 }
```

Listing 3.2: Schema to widget factory (excerpt)

```
1 elif field_type in ["line", "number"]:
2     widget = QLineEdit()
3     widget.setText(str(field_def.get("default", "")))
4     v = field_def.get("validator")
5     if v and v["type"] == "double_range":
6         widget.setValidator(QDoubleValidator(v["bottom"], v["top"], v["
7             decimals"]))
8         widget.setPlaceholderText(f"{v['bottom']} - {v['top']}")
9     bind = field_def.get("bind")
10    if bind:
11        setattr(self, bind, widget)      # accessible as dialog.bind_name
```

Listing 3.3: Support Conditions tab: CAD preview wiring

```
1 self.left_cad = SupportCADWidget()      # girder + supports
2 self.right_cad = SupportDetailCADWidget() # bearing detail
3 self.parent_dialog.bearing_length_input.textChanged.connect(self.
4     update_cad)
5 def update_cad(self):
6     try:
7         value = float(self.parent_dialog.bearing_length_input.text())
8     except ValueError:
9         value = 400
```

```
10 self.right_cad.update_params({"bearing_length": value})
```

Listing 3.4: Save: validate all tabs, single pop-up

```
1 errors = []
2 for tab in [self.support_tab, self.design_options_tab,
3             self.design_options_cont_tab]:
4     errors.extend(tab.validate_tab() or [])
5
6 if errors:
7     self._show_validation_errors(errors)    # one popup, all messages
8     return
9
10 for tab in tabs:
11     saved.update(tab.save_values() or {})
```

3.3.3 Explanation of the Code

- **Listing 3.1:** a field is fully described by data, so adding or changing a field needs no GUI code.
- **Listing 3.2, lines 1–7:** the factory creates a `QLineEdit`, sets the default and attaches a range validator with a placeholder showing the limits; **lines 9–11** bind the widget onto the dialog so tabs can reach it by name.
- **Listing 3.3, lines 1–3:** two CAD widgets are placed side by side and the bearing-length field is connected to `update_cad`; **lines 5–10** parse the value (falling back to 400) and redraw the bearing detail.
- **Listing 3.4, lines 1–4:** every tab reports its own errors; **lines 6–8** show them together and abort; **lines 10–11** merge each tab's values into the saved dictionary.

Chapter 4

Generate Results Dialog

4.1 Task 2: Problem Statement

After a design run, OsdagBridge stores a large number of intermediate and final quantities (bridge configuration, material and section properties, load effects, deflections, stresses, ULS/SLS/fatigue checks, shear-connector design, and the design summary). There was no way for the user to browse these results in the application or to take them out of it. The Output dock had a “Generate Results” button that only showed a placeholder message.

The task was to build the user interface for a *Generate Results* dialog that lets the user

- see every available result table organised in a hierarchy,
- pick any subset of tables,
- preview the selected tables inside the dialog, and
- export the chosen tables to a formatted Excel workbook.

Only the UI and the wiring to existing input/output data were in scope; the design calculations themselves were not part of this task.

4.2 Task 2: Tasks Done

4.2.1 Table hierarchy

All result tables are declared as one nested dictionary (`GENERATE_RESULTS_DEFAULTS`) in `ui_fields_additional_input.py`. It has three levels: main category → group → table, and every table entry carries an `id`, a display `label`, its column headers and placeholder rows. The dialog reads this dictionary to build its tree, so a new table can be added by editing the dictionary alone. Table 4.1 summarises the hierarchy.

Main category	Group	Tables
Model Definition	Bridge Configuration	Summary, Material Properties (Steel, Concrete)
	Load Definitions	Permanent, Live, Wind, Seismic, Temperature, Load Combinations
	Member Definitions	Girder, Cross Bracing, End Diaphragm, Shear Stud, Deck Slab
Analysis Results	Load Effects – Girder	BM/SF envelopes and by load case
	Deflections	Live Load, Total Load
	Stress Results	Steel, Concrete Deck, Reinforcement (service)
Design Results	ULS Checks	Flexure, Shear, Bending–Shear Interaction, LTB (construction)
	SLS Checks	Deflection Control, Stress Limitation (Steel, Concrete, Reinforcement)
	Fatigue Checks	Girder, Shear Connectors
	Shear Connector Design	Capacity, Spacing (ULS, Fatigue), Governing Spacing, Detailing
	Transverse and Crack Checks	Transverse Shear in Slab, Crack Width
	Design Summary	Design Results Summary

Table 4.1: Result tables available in the Generate Results dialog.

4.2.2 Dialog design

The dialog (`GenerateResultsDialog`) is a frameless window with the application’s custom title bar, and holds two pages in a stacked widget (Figure 4.1).

1. **Generate Results page.** A three-level tree of check boxes built from the hierarchy above. Ticking a group ticks all of its tables, and a group shows a partial state when only some children are ticked. “Select All” / “Clear All” buttons act on the whole tree. Clicking a table name shows its description. “Show Selections” moves to the second page; if nothing is selected a warning pop-up is shown.

2. **Export Table page.** The left panel repeats the tree, but restricted to the tables chosen on page 1, and can be collapsed to give the preview more room. The right panel is a table preview: clicking a tree entry loads that table with its headers and rows. The columns are auto-fitted on resize. “Back” returns to page 1 keeping the selection; “Export” writes an Excel file.

Figure 4.1: Generate Results page: hierarchical selection of result tables.

Figure 4.2: Export Table page: collapsible tree on the left, table preview on the right.

4.2.3 Custom widgets

A few small widgets were written to match the Osdag look and feel: a checkbox delegate that paints the tree’s check boxes in the Osdag green, a combo box that ignores mouse-wheel events inside scroll areas, a rounded frame that can be greyed out while the tree panel is collapsed, and a vertical label used as the collapsed panel’s caption.

4.2.4 Populating the tables with user values

The placeholder rows are replaced with live values through a values builder (`generate_results_values.L`). Each table `id` is mapped to a resolver function that reads the merged input dictionary (main input dock + additional inputs) and the current bridge object, converts units (Pa to MPa, m to mm, etc.) and returns the rows. Tables without a resolver keep their placeholder rows, so the UI works even before all calculations exist. When the user opens the dialog, the defaults are deep-copied and each resolvable table is overlaid with live data.

4.2.5 Excel export

The Export button asks for a file name and writes every checked table to one workbook using `openpyxl`. Each table gets a title row, a green header row with white bold text, alternating row shading and thin green borders, and column widths fitted to the content. If no table is checked a warning is shown instead.

4.2.6 Integration with the Output dock

The existing “Generate Results” button in the Output dock now opens the dialog. If no design has been run yet, a “No design created!” warning is shown instead of opening it.

4.3 Task 2: Documentation

4.3.1 Files

- `desktop/ui/dialogs/generate_results_dialog.py` – `GenerateResultsPage`, `ExportTablePage`, `GenerateResultsDialog` and the helper widgets.
- `desktop/ui/dialogs/generate_results_values_builder.py` – `resolve_table()` and one `resolve_*` function per table, plus unit-conversion helpers.
- `core/bridge_types/plate_girder/ui_fields_additional_input.py` – `GENERATE_RESULTS_DEFAULTS` (table hierarchy, headers, placeholder rows, descriptions).
- `desktop/ui/docks/output_dock.py` – `open_generate_results_dialog()` hooked to the button.

4.3.2 User workflow

1. Run a design, then click **Generate Results** in the Output dock.
2. Tick the required tables or groups (or **Select All**) and click **Show Selections**.
3. Click any table in the left tree to preview it; untick tables that are not needed.
4. Click **Export**, choose a file name, and the workbook is saved.

4.3.3 Adding a new result table

1. Add an entry under the appropriate group in `GENERATE_RESULTS_DEFAULTS` with a unique `id`, `label`, `headers`, `placeholder rows` and a `description`. The tree, preview and export pick it up automatically.

2. Optionally write a `resolve_<id>()` function in the values builder and register it in the resolver map, so the table shows live values instead of placeholders.

Chapter 5

Conclusions

5.1 Tasks Accomplished

During the fellowship two main contributions were made to the OsdagBridge plate-girder module.

1. **Schema-driven Additional Inputs tabs.** The Support Conditions, Analysis/Design Options and Design Options (Cont.) tabs were separated into their own files and rebuilt around a common schema. Every field – its label, widget type, default value and permissible range – is now declared once in a Python dictionary and rendered by a shared widget factory in the parent dialog. Each tab implements uniform `validate_tab()`, `save_values()` and `reset_defaults()` hooks, so the Save button validates all tabs and reports every error in a single pop-up, and the Defaults button restores schema values reliably. The Support Conditions tab gained a live CAD preview that redraws as the bearing length is edited, and a reinforcement “Set Bounds” dialog was added to the Deck Design card.
2. **Generate Results dialog.** A two-page dialog was designed and implemented to browse and export design results. The first page shows a three-level tree of all result tables (model definition, analysis results and design checks) with tri-state check boxes; the second page previews any selected table and exports the chosen set to a formatted Excel workbook. Table definitions live in one dictionary, a values builder maps table identifiers to resolver functions that fill them with the user’s inputs and computed quantities, and the dialog is wired to the existing Generate

Results button in the Output dock.

Both pieces follow the same principle: keep data definitions separate from GUI code so that new fields or tables can be added without touching the widget logic. This reduced duplicated code, made the dialogs easier to extend, and gave the OsdagBridge team a consistent pattern to follow for the remaining tabs.

5.2 Skills Developed

Technical skills

- **PySide6 / Qt desktop development:** building complex dialogs with stacked pages, tab widgets, tree views with tri-state check boxes, custom delegates and painted widgets, scroll areas and stylesheet-based theming.
- **Schema-driven UI design:** describing forms as data and generating widgets, validators and defaults from it; designing small, uniform interfaces (`validate_tab`, `save_values`) between a parent dialog and its tabs.
- **Input validation and error reporting:** combining Qt validators with explicit range and cross-field checks, and collecting errors into a single user-facing message.
- **Data export and unit handling:** generating styled Excel workbooks with `openpyxl`, and writing resolver functions that convert SI quantities (Pa, m, m⁴) into engineering units for display.
- **Working in a large codebase:** reading existing Osdag modules, reusing shared components (custom message boxes, title bars, CAD widgets) and matching the project's coding conventions.
- **Version control:** working on feature branches, rebasing on a moving `dev` branch, resolving merge conflicts and preparing pull requests for review.

Professional skills

- Translating requirement documents and mentor feedback into concrete UI behaviour, and iterating on it through review cycles.

- Communicating design decisions and progress clearly in a distributed open-source team.
- Breaking a large feature into small, testable increments and prioritising work under time constraints.
- Writing developer-facing documentation so that others can extend the work after the fellowship.

Chapter A

Appendix

A.1 Work Reports

Name: Jatin Nainwani **Project:** OsdagBridge (Plate Girder Module) **Internship:** FOSSEE Fellowship 2026

#	Date	Day	Task	Hours
25	26-Mar-2026	Thursday	Placed both CAD widgets side by side in the Support Conditions tab; sizing and resize behaviour	5
26	27-Mar-2026	Friday	Tested tab on Windows with different window sizes; fixed scroll issues	4
27	30-Mar-2026	Monday	Rebased on latest dev; resolved merge conflicts in tabs and CAD files	6
28	31-Mar-2026	Tuesday	Regression-tested the dialog after rebase	4
29	01-Apr-2026	Wednesday	Fixed remaining merge conflicts for the Additional Inputs tab and schema	5
30	02-Apr-2026	Thursday	Cleaned up imports and unused code; prepared pull request	4
31	03-Apr-2026	Friday	Addressed PR review comments	4
32	06-Apr-2026	Monday	Compared UI with requirement document; minor spacing and alignment fixes	4
33	07-Apr-2026	Tuesday	Tested Defaults/Save with locked and unlocked dialog states	4
34	08-Apr-2026	Wednesday	Discussed file organisation of CAD widgets with mentor	3
35	09-Apr-2026	Thursday	Moved CAD files into tabs/drawings/ and updated imports	4
36	10-Apr-2026	Friday	Final testing of Task 1; pushed restructured branch	5
37	13-Apr-2026	Monday	Began Task 2: studied designer.py and plategirderbridge.py to understand the design workflow	5
38	14-Apr-2026	Tuesday	Mapped UI inputs to the internal input dictionary for Plate Girder	6
39	15-Apr-2026	Wednesday	Updated plategirderbridge.py with new demand envelope logic	6
40	16-Apr-2026	Thursday	Integrated IRC 6 (Table B.2) load combinations into the design code	5
41	17-Apr-2026	Friday	Modified demand envelope generation to match designer.py implementation	6
42	20-Apr-2026	Monday	Debugged output dictionary and results schema generation	5
43	21-Apr-2026	Tuesday	Worked on DCR output formatting and result display	5
44	22-Apr-2026	Wednesday	Discussed UI changes for results tab with mentor	4
45	23-Apr-2026	Thursday	Implemented percentage bar widget for DCR values	5
46	24-Apr-2026	Friday	Fixed issues with bar display and removed numbering column	4
47	27-Apr-2026	Monday	Worked on report generation: LaTeX templates and rendering	5
48	28-Apr-2026	Tuesday	Final testing of Task 2; documentation and cleanup	5
Total Hours				228

Name: Jatin Nainwani **Project:** OsdagBridge (Plate Girder Module) **Internship:** FOSSEE Fellowship 2026

#	Date	Day	Task	Hours
1	23-Feb-2026	Monday	Set up conda environment, cloned OsdagBridge and ran the application; read contribution guidelines	4
2	24-Feb-2026	Tuesday	Studied the plate-girder input dock and the existing Additional Inputs dialog code	5
3	25-Feb-2026	Wednesday	Went through the requirement PDF for Support Conditions and Design Options fields; listed required inputs and ranges	5
4	26-Feb-2026	Thursday	Discussed with mentor the plan to split the dialog into separate tab files driven by a schema	4
5	27-Feb-2026	Friday	Designed the schema format (label, type, default, validator, bind) and drafted the widget factory	6
6	28-Feb-2026	Saturday	Created design_options_tab.py, design_options_cont_tab.py and support_conditions_tab.py; added schema ranges and defaults	6
7	02-Mar-2026	Monday	Wired the new tabs into the dialog; fixed the Save button not working	5
8	03-Mar-2026	Tuesday	Replaced per-field pop-ups with a single validation popup listing all errors	6
9	04-Mar-2026	Wednesday	Removed unnecessary legacy code; moved defaults into the schema and validation into each tab	6
10	05-Mar-2026	Thursday	Fixed the "coming soon" error on the Defaults button; created separate cards for Support Conditions and Bearing Length	5
11	06-Mar-2026	Friday	Added combobox_utils.py for greyed-out disabled combo items with cursor feedback	4
12	09-Mar-2026	Monday	Made all default values display with two decimals; enforced decimal places in validators	5
13	10-Mar-2026	Tuesday	Fixed inputs not being retained after Save; tested Save/Defaults on all three tabs	5
14	11-Mar-2026	Wednesday	Implemented Limit States checkbox groups (ULS / SLS) in Design Options (Cont.)	6
15	12-Mar-2026	Thursday	Added Deflection Control row layout and Partial Factor fields with IRC 6 ranges	5
16	13-Mar-2026	Friday	Weekly review with mentor; noted UI feedback on labels and card layout	3
17	16-Mar-2026	Monday	Applied label and layout changes from review; added Description Box on Design Options tab	5
18	17-Mar-2026	Tuesday	Implemented Reinforcement Size "Set Bounds" dialog with 8–40 mm limits	6
19	18-Mar-2026	Wednesday	Added shear stud spacing check (spacing $\geq 4 \times$ diameter) and tested edge cases	5
20	19-Mar-2026	Thursday	Studied existing CAD drawing widgets in Osdag to reuse for support conditions	4
21	20-Mar-2026	Friday	Started support_conditions_cad.py: girder line with support symbols	6
22	23-Mar-2026	Monday	Continued support CAD: pinned/roller symbols, dimensions and scaling	6
23	24-Mar-2026	Tuesday	Started support_detail_cad.py for the bearing detail drawing	6
24	25-Mar-2026	Wednesday	Connected bearing length input to live CAD update; handled invalid input fallback	5

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.