FOSSEE Winter Internship Report

On

Development of Desktop App for Osdag

Submitted by

Jatin Nainwani

3rd Year B.Tech Student, 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 senior 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

Chapter 1

Introduction

1.1 National Mission in Education through ICT

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

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

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

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

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

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

1.1.1 ICT Initiatives of MoE

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

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

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

1.2 FOSSEE Project

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

1.2.1 Projects and Activities

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

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

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

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

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

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

1.3.1 Osdag GUI

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

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

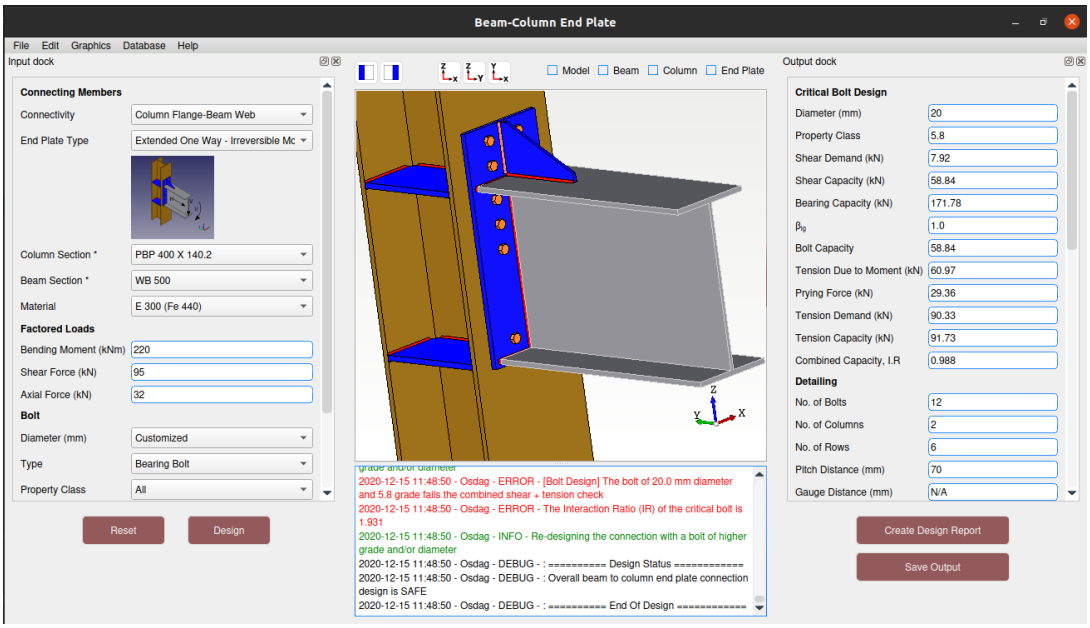


Figure 1.2: Osdag GUI

1.3.2 Features

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

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

Not Applicable - Internship continued.

2.2 Tasks Done

Not Applicable - Internship continued.

Chapter 3

Refactoring of the Plate Girder Designer and Deflection Pipeline

3.1 Task 1: Problem Statement

The plate girder design module of Osdag Bridge (`designer.py`) is the Stage-5 engine that converts grillage analysis results into IRC 22:2015 demand-to-capacity (DCR) checks. Before this task the module had grown into a single, tightly coupled file with several structural problems:

- **Duplicated equations.** The steel I-section properties (A, $\bar{y}$, I_z, Z_e, Z_p) were computed independently in `SteelSection` (designer) and again in `BridgeConfigurationSolver` (initial sizing), with slightly different formulae. Any change to the girder shape had to be made in two places.
- **Silent hard-coded defaults.** Stud diameter, stud height, fatigue cycles N_{sc}, bottom cover, partial safety factors and the shear-buckling method all fell back to literal numbers (e.g. `or 22.0`, `Nsc = 2_000_000`) when a user input was missing, so the design could run on values the user never saw.
- **Deck checks inside the steel engine.** Concrete stress, rebar stress, crack control (Cl. 604.4) and transverse shear (Cl. 606.10) were evaluated in the steel DCR engine using an assumed slab reinforcement, although the real reinforcement is only known after deck design (Stage 6).

- **Magic strings.** Composite-section results were passed around as literal dictionary keys (`"I_comp_mm4"`, `"y_top_mm"`), which made typos undetectable.
- **Fragile girder identification.** Girders were rebuilt from the analysis dataset using an x-coordinate equality test that collapses on skew bridges; edge beams had to be filtered out by name in several places.
- **No camber.** Deflections were reported on the bare-steel section and fabrication camber – an input already present in the "Design Options (Cont.)" tab – was never used in the deflection check.

The task was therefore to refactor the designer and its related files so that (i) every equation has a single source, (ii) every design input is explicit and validated, (iii) deck-only checks move to the deck design stage, (iv) girder data comes from the post-processed result set, and (v) camber is applied consistently to the deflection checks and the report.

3.2 Task 1: Tasks Done

The work was delivered as six commits on the `Gen_res_refactor` branch (Table 3.1).

Table 3.1: Commits delivered for Task 1

Commit	Date	Summary
`f9d32bc`	09 Jul 2026	Refactored designer and deck design; moved deck checks; single-source section properties; keyed composite dictionary; Design Envelope engine
`c9704bc`	05 Aug 2026	Designer reads girders from `result_data` (post-processed, skew-safe)
`5f2729e`	14 Aug 2026	Camber added to deflection calculations; new DL deflection check #18
`8c9152a`	14 Aug 2026	Deflection check shown even when camber cancels the sag to zero
`333f1e5`	17 Aug 2026	Deflection cache carries both raw (pre-camber) and post-camber values
`bb779b2`	19 Aug 2026	Raw deflections keyed by `KEY_SD_DEFL_*_RAW` constants end-to-end

3.2.1 Single source for steel section properties

A new pure function `steel_i_section_properties()` was added to `initial_sizing.py`. It is unit-agnostic and returns a dictionary keyed by the member-property constants

in common.py (KEY_MP_GIRDER_SECTIONAL_IZ, KEY_MP_GIRDER_PLASTIC_MODULUS_ZUZ, ...). Both consumers now only *read* from it:

- BridgeConfigurationSolver.compute_section_properties() (initial sizing, metres) – the symmetric and unsymmetric branches lost ∼100 lines of duplicated algebra.
- SteelSection.__post_init__() (designer, mm) – the six @property methods were replaced by one call at construction.

Four new keys were added to common.py so the whole vocabulary is complete: KEY_MP_GIRDER_CENTR , _FLANGE_AREA_TOP, _FLANGE_AREA_BOT, _WEB_AREA. Weak-axis, torsion (I_t) and warping (I_w) constants were also folded into the same function.

3.2.2 Keyed composite-section dictionary

Eight constants KEY_COMP_N, KEY_COMP_I, KEY_COMP_Y_TOP, KEY_COMP_Y_BOT, KEY_COMP_S_TOP , KEY_COMP_S_BOT, KEY_COMP_Y_FROM_BOT and KEY_COMP_AC_TRANS_MM2 replaced every literal string used to address the output of composite_section_properties() in the designer, the report generator and the demand extractor.

3.2.3 Removal of silent defaults

All input dataclasses in Section 1 of the designer were made strict. Table 3.2 lists the fields whose hard-coded fallback was removed and the UI key that now supplies them through the _req() validator, which raises a ValueError naming the missing key.

Table 3.2: Hard-coded defaults removed in BridgeConfig.from_plate_girder_bridge()

Dataclass field	Old fallback	Now read from
SteelProperties.fck/fctm/Ecm	IRC 22 Annex III table	KEY_MATERIAL_DECK_FCK/FCTM/EC
SteelProperties.rebar_grade	"Fe500"	KEY_DS_REINF_MATERIAL
SteelProperties.Es/Gs/nu	module constants	material DB / KEY_MATERIAL_GIR
SteelProperties.gamma_*	1.10 / 1.25 / 1.25	KEY_DO_GAMMA_M0/M1/V/MF
SlabProperties.cover_bot	25 mm	KEY_DS_BOTTOM_CLEAR_COVER
ShearStudConfig.*	22/150/500/350/2/100	KEY_DS_STUD_*
FatigueConfig.Nsc	2×10^6	KEY_DO_LOAD_CYCLES
StiffenerConfig.shear_method	"post_critical"	KEY_MP_STIFFENER_DESIGN_METHO
GeometryConfig.beam_type	"inner"	derived from girder index
GeometryConfig.support_type	"simply_supported"	KEY_SC_LEFT/RIGHT_SUPPORT
GeometryConfig.camber_mode/value	not read	KEY_DO_CAMBER_MODE/VALUE

The obsolete SteelProperties.from_grades() and BridgeConfig.example_33m_bridge() constructors were deleted.

3.2.4 Deck-only checks moved to deck design

Four checks that depend on the slab reinforcement were removed from IRC22CapacityCalculator and DCREngine and re-implemented in deckdesign.py as module-level functions crack_control_As_mi() and transverse_shear_check(). design_deck_slab() gained three keyword arguments – design_results, bf_top_mm, stud_height_mm – so that, after the slab bars are chosen, it runs the composite interface checks with the *designed* A_s and writes the values back into design_results under the KEY_SD_TS_* / KEY_SD_CRACK_* keys for the report tables and the deck dialog utilisation bars.

Table 3.3 shows the resulting re-mapping of check IDs in the eight-category aggregation used by the Output Dock and the Steel Design check tab.

Table 3.3: Check-ID mapping before and after the refactor

Category card	Before	After
Resistance to Longitudinal (and Transverse) Shear	6, 7, 16, 17	6, 7
Stress Limitation	10, 11, 12	11
Deflection (and Crack Control)	13, 14, 15	13, 14, 18
Deck design (new home)	–	10, 12, 15, 16, 17

Deck design itself was strengthened at the same time: the required-steel quadratic was corrected to be the exact inverse of the capacity formula (the 0.42 lever-arm fac-

tor was missing), a 12 mm practical minimum bar was introduced, and two iterative helpers `_tighten_for_sls_stress()` and `_tighten_for_oneway_shear()` increase reinforcement until SLS steel stress and one-way shear reach a 0.95 target utilisation. A small reusable geometry module `deck_reinforcement/geometry.py` (`bar_area_mm2`, `reinforcement_area_per_m_mm2`) was created.

3.2.5 Design Envelope as a synthesised engine

The "Design Envelope" pseudo load case was removed from the Steel Design dialog and rebuilt as `design_envelope_engine()`. Instead of synthesising a fictitious demand, it takes the already-computed per-load-case check results, keeps the highest DCR per check ID, and returns the girder whose worst check governs. It is exposed only in the Output Dock's design load-combination dropdown, never in the analysis dropdown.

3.2.6 Girders from post-processed `result_data`

`PlateGirderBridge.result_data` is now built *before* Stage 5 and passed into `run_design_check()`. The demand extractor reads `result_data["girders"]` (labelled G1…Gn, edge beams already excluded, skew-safe), so all `if g_name in ("EB1","EB2")` filters were deleted. The Steel Design dialog resolves the selected girder from the combo index (`dx+1f "Gi"`) rather than the raw analysis key.

3.2.7 Composite-basis deflections and camber

The grillage is modelled on bare steel, so every reported displacement must be divided by the stiffness ratio I_{comp}/I_{steel}. This correction was moved to one place, `composite_stiffness_props()` in `results_data.py`, and `build_deflections_cache()` was rewritten to:

- take live-load deflection as the maximum over *every* vehicle position (the single "1.0 LL" case is the maximum-moment position, not the maximum-sag position);
- add the DL-only case and a per-load-case dictionary;
- apply the composite correction to all of them, and return values under `KEY_SD_DEFL_LIVE_RAW`, `KEY_SD_DEFL_TOTAL_RAW`, `KEY_SD_DEFL_DL_RAW`.

Camber is a design decision, so it lives in the designer: `apply_camber()` subtracts the camber (full DL sag in Default mode, user value in Custom mode, capped at `KEY_MAX_CAMBER_M` = 4 m) from DL and DL+LL sags, clamped at zero, and `apply_camber_to_deflections_cache()` produces a cache that carries *both* the raw and post-camber values. A new check #18 "SLS Deflection (DL)" (post-camber, limit $L/600$) was added, and checks 14/18 are gated on the limit rather than on the value so that a 0 mm result after camber is still displayed. Report Chapter 4 (analysis results) reads the `*_RAW` keys so its table matches the deflection plots, while Chapter 5 (design) reads the post-camber keys `KEY_SD_DEFL_AFTER_CAMBER` and `KEY_SD_APPLIED_CAMBER`.

Figure 3.1: Data flow of the refactored deflection pipeline: OpenSees displacements → `build_deflections_cache` (composite correction) → `apply_camber_to_deflections_cache` (camber) → `DCREngine` checks 13/14/18 and report Chapters 4/5.

Figure 3.2: Steel Design check tab after the refactor, showing the "Deflection" card with live, total and DL (post-camber) checks.

3.3 Task 1: Python Code

This section presents the key code fragments developed for the refactor. The full implementations live in `core/bridge_types/plate_girder/designer.py`, `initial_sizing.py`, `results_data.py` and `deckdesign.py`.

3.3.1 Description of the Scripts

- **Section properties**: one unit-agnostic function returning a keyed dictionary consumed by both initial sizing and the designer.
- **Input validation**: a small `_req()` helper that turns a missing UI value into a descriptive error instead of a silent default.
- **Deck interface checks**: crack control and transverse shear run in `design_deck_slab()` with the designed reinforcement.
- **Design Envelope**: a synthetic `DCREngine` built from per-load-case check results.

- **Deflection and camber**: composite correction in the results layer, camber in the design layer, raw and adjusted values both preserved.

3.3.2 Python Code

Listing 3.1: Single source of steel I-section equations (`initial_sizing.py`)

```python
def steel_i_section_properties(D, bf_top, tf_top, bf_bot, tf_bot, tw)
    -> dict:
    """Bare-steel section properties of a welded I-girder. Unit-
        agnostic:
    callers must *read* these values, never recompute them."""
    dw     = D - tf_top - tf_bot
    Af_top, Af_bot, Aw = bf_top * tf_top, bf_bot * tf_bot, dw * tw
    A_steel = Af_top + Aw + Af_bot

    y_b, y_w, y_t = tf_bot / 2, tf_bot + dw / 2, tf_bot + dw + tf_top /
        2
    y_cg = (Af_bot * y_b + Aw * y_w + Af_top * y_t) / A_steel

    Iz = (bf_bot * tf_bot**3 / 12 + Af_bot * (y_cg - y_b)**2
          + tw * dw**3 / 12       + Aw     * (y_cg - y_w)**2
          + bf_top * tf_top**3 / 12 + Af_top * (y_cg - y_t)**2)
    ...
    return {
        KEY_MP_GIRDER_SECTIONAL_AREA:      A_steel,
        KEY_MP_GIRDER_SECTIONAL_IZ:        Iz,
        KEY_MP_GIRDER_ELASTIC_MODULUS_ZZ:  Iz / max(y_cg, D - y_cg),
        KEY_MP_GIRDER_PLASTIC_MODULUS_ZUZ: Zp,
        KEY_MP_GIRDER_CENTROID_YCG:        y_cg,
        KEY_MP_GIRDER_WEB_DEPTH:           dw,
        ...
    }
```

Listing 3.2: `SteelSection` now reads from the shared function (`designer.py`)

```python
@dataclass
class SteelSection:
    D: float; bf_top: float; tf_top: float
    bf_bot: float; tf_bot: float; tw: float
```

```
5
6       def __post_init__(self) -> None:
7           props = steel_i_section_properties(
8               D=self.D, bf_top=self.bf_top, tf_top=self.tf_top,
9               bf_bot=self.bf_bot, tf_bot=self.tf_bot, tw=self.tw)
10          self.dw       = props[KEY_MP_GIRDER_WEB_DEPTH]
11          self.A_steel  = props[KEY_MP_GIRDER_SECTIONAL_AREA]
12          self.Iz_steel = props[KEY_MP_GIRDER_SECTIONAL_IZ]
13          self.Zp_steel = props[KEY_MP_GIRDER_PLASTIC_MODULUS_ZUZ]
14          self.Ze_steel = props[KEY_MP_GIRDER_ELASTIC_MODULUS_ZZ]
15          self.y_cg_from_bot = props[KEY_MP_GIRDER_CENTROID_YCG]
```

Listing 3.3: Required inputs – no silent fallbacks (`designer.py`)

```
1  def _req(value, key, source):
2      if value is None or (isinstance(value, str) and not value.strip()):
3          raise ValueError(f"Required input {key!r} is missing from {
               source}.")
4      return value
5
6  # in BridgeConfig.from_plate_girder_bridge():
7  ai = bridge.additional_inputs
8  studs = ShearStudConfig(
9      diameter = float(_req(ai.get(KEY_DS_STUD_DIAMETER),
           KEY_DS_STUD_DIAMETER, "additional_inputs")),
10     height   = float(_req(ai.get(KEY_DS_STUD_HEIGHT),
           KEY_DS_STUD_HEIGHT,   "additional_inputs")),
11     ...)
12 fatigue = FatigueConfig(
13     Nsc = int(float(_req(ai.get(KEY_DO_LOAD_CYCLES), KEY_DO_LOAD_CYCLES
           , "additional_inputs"))))
14
15 # beam_type is a position property, not a user input
16 n_main = bridge._girder_count()
17 beam_type = "outer" if girder_index in (0, n_main - 1) else "inner"
```

Listing 3.4: Composite interface checks moved into deck design (`deckdesign.py`)

```
1  def design_deck_slab(input_dict, fck, fctm, fy, Es, Ecm, *,
2                       design_results=None, bf_top_mm=0.0, stud_height_mm
```

```
                          =0.0):
    ...   # size bottom / top / overhang bars (ULS, min steel, SLS, one
        -way shear)

    # 10c. composite steel-concrete interface checks (IRC 22:2015)
    dr = design_results
    if dr and dr.get("beff_mm") and dr.get("VL_N_per_mm") is not None:
        ts = transverse_shear_check(                # Cl.606.10
            VL_N_per_mm=dr["VL_N_per_mm"], fck_MPa=fck, fy_rebar_MPa=fy
                ,
            bf_top_mm=bf_top_mm, stud_height_mm=stud_height_mm,
            t_slab_mm=deck_t_mm, As_total_mm2=As_bot + As_top)
        crack = crack_control_As_min(               # Cl.604.4
            beff_mm=dr["beff_mm"], fctm_MPa=fctm, fy_rebar_MPa=fy,
            t_slab_mm=deck_t_mm, As_total_mm2=As_top * dr["beff_mm"] /
                1000.0)

        dr[KEY_SD_TS_VL]        = ts["VL_N_per_mm"]
        dr[KEY_SD_TS_VRD]       = ts["governing_capacity_kN_per_m"]
        dr[KEY_SD_CRACK_AS_MIN] = crack["As_min_mm2"]
        dr[KEY_SD_CRACK_AS_PROV]= crack["As_provided_mm2"]
```

Listing 3.5: Design Envelope synthesised from per-load-case results (`designer.py`)

```
def design_envelope_engine(girder_names, per_girder):
    best_engine, best_max = None, -1.0
    for g_name in girder_names:
        worst_by_id: dict[int, CheckResult] = {}
        for lc_res in per_girder[g_name]["per_lc"].values():
            for chk in lc_res.get("checks", []):
                cid, dcr = chk["id"], float(chk.get("dcr", 0.0))
                if cid not in worst_by_id or dcr > worst_by_id[cid].dcr
                    :
                    worst_by_id[cid] = CheckResult(check_id=cid, dcr=
                        dcr, ...)
        engine = DCREngine.__new__(DCREngine)       # synthetic, no
            demand/capacity
        engine.checks = list(worst_by_id.values())
        if max(c.dcr for c in engine.checks) > best_max:
            best_engine, best_max = engine, max(c.dcr for c in engine.
```

```
                checks)
    return best_engine
```

Listing 3.6: Composite-basis deflection cache (`results_data.py`)

```
def composite_stiffness_props(config) -> tuple[dict, float]:
    """Short-term composite props + I_comp / I_steel (floored at 1.0)."""
    props = composite_section_properties(...)
    return props, max(props[KEY_COMP_I] / config.section.Iz_steel, 1.0)

def build_deflections_cache(result_handler, config, result_data) -> dict:
    live_lcs = result_handler.classify_loadcases()["vehicle_static"]  # every vehicle position
    _, stiffness_ratio = composite_stiffness_props(config)

    def _max_defl_mm(lc, nodes):
        vals = [abs(float(disp_da.sel(Loadcase=lc, Node=n, Component="y"))) * 1000.0
                for n in nodes if n in node_set]
        return round(max(vals) / stiffness_ratio, 3) if vals else None

    cache = {}
    for label, g in result_data["girders"].items():      # G1..Gn, skew-safe
        nodes = g["nodes"]
        cache[label] = {
            KEY_SD_DEFL_LIVE_RAW:  max(_max_defl_mm(lc, nodes) for lc in live_lcs),
            KEY_SD_DEFL_TOTAL_RAW: _max_defl_mm(dl_ll_case, nodes),
            KEY_SD_DEFL_DL_RAW:    _max_defl_mm(dl_case, nodes),
            "per_lc":              _per_lc_defl(nodes),
        }
    return cache
```

Listing 3.7: Camber applied in the design layer (`designer.py`)

```
MAX_CAMBER_MM = KEY_MAX_CAMBER_M * 1000.0          # 4 m buildable limit
```

```
2
3  def apply_camber(girder_defl, camber_mode, camber_value_m):
4      dl    = float(girder_defl[KEY_SD_DEFL_DL_RAW])
5      total = float(girder_defl[KEY_SD_DEFL_TOTAL_RAW])
6      camber = max(dl, 0.0) if camber_mode.lower() == "default" \
7               else max(float(camber_value_m) * 1000.0, 0.0)
8      camber = min(camber, MAX_CAMBER_MM)
9      return max(dl - camber, 0.0), max(total - camber, 0.0), camber
10
11 def apply_camber_to_deflections_cache(raw_cache, camber_mode,
       camber_value_m):
12     out = {}
13     for label, d in raw_cache.items():
14         dl_adj, total_adj, camber_mm = apply_camber(d, camber_mode,
               camber_value_m)
15         out[label] = {
16             "live_mm":   d.get(KEY_SD_DEFL_LIVE_RAW),   # live is
                   uncambered
17             "dl_mm":     round(dl_adj, 3),
18             "total_mm":  round(total_adj, 3),
19             "camber_mm": round(camber_mm, 3),
20             "per_lc":    d.get("per_lc", {}),
21             KEY_SD_DEFL_LIVE_RAW:  d.get(KEY_SD_DEFL_LIVE_RAW),   # pre
                   -camber, for report Ch.4
22             KEY_SD_DEFL_TOTAL_RAW: d.get(KEY_SD_DEFL_TOTAL_RAW),
23             KEY_SD_DEFL_DL_RAW:    d.get(KEY_SD_DEFL_DL_RAW),
24         }
25     return out
```

Listing 3.8: New DL deflection check and limit-gated total check (DCREngine)

```
1  # Total deflection: gated on the LIMIT, not the value -- a large camber
       clamps
2  # the sag to 0 mm, and 0 is a real result that must still be shown.
3  if _dl_ll and c.defl_limit_total_mm > 0:
4      self._add_check(14, "SLS Deflection (Total)", "Cl.604.3.2",
5                      d.delta_total_mm, c.defl_limit_total_mm, "mm", note
                           ="Limit = L/600")
6
7  # DL deflection (post-camber): only for the DL-only case.
```

```
if _dl_only and c.defl_limit_total_mm > 0:
    note = (f"Limit = L/600 | camber = {d.camber_mm:.1f} mm"
            if d.camber_mm > 0 else "Limit = L/600")
    self._add_check(18, "SLS Deflection (DL)", "Cl.604.3.2",
                    d.delta_dl_mm, c.defl_limit_total_mm, "mm", note=
                        note)
```

Listing 3.9: Pipeline wiring in `run_design_check()`

```
result_data = getattr(plate_girder_bridge, "result_data", None)
if not (result_data and result_data.get("girders")):
    raise ValueError("plate_girder_bridge.result_data must contain '
        girders' ...")

if deflections_cache is None:
    deflections_cache = apply_camber_to_deflections_cache(
        build_deflections_cache(analysis_results, config, result_data),
        config.geometry.camber_mode, config.geometry.camber_value_m)
    plate_girder_bridge._deflections_cache = deflections_cache

per_girder_demands, per_girder_per_lc =
    _extract_demands_from_analysis_results(
    analysis_results, config, deflections_cache, result_data)
```

3.3.3 Explanation of the Code

- **Listing 3.1–3.2**: the section equations exist once. `SteelSection` computes its properties at construction by calling the shared function and stores them as plain attributes; the outputs are addressed by `KEY_MP_GIRDER_*` constants so initial sizing, the designer and the report speak the same vocabulary.

- **Listing 3.3**: `_req()` converts a missing UI value into a `ValueError` that names the key and its source dictionary. `beam_type` and `support_type` are derived from the girder position and the support-condition inputs instead of being defaulted.

- **Listing 3.4**: `design_deck_slab()` receives the Stage-5 `design_results` dictionary. After the slab reinforcement is finalised it runs the IRC 22 interface checks

with the real A_s and mutates `design_results` in place, so the report and the deck dialog read the same numbers.

- **Listing 3.5**: the envelope is built from already-computed checks, so it costs no extra analysis. Because each per-LC engine only emits the checks its load-case type influences, the per-ID maximum is automatically "worst utilisation over the load cases that affect that check".

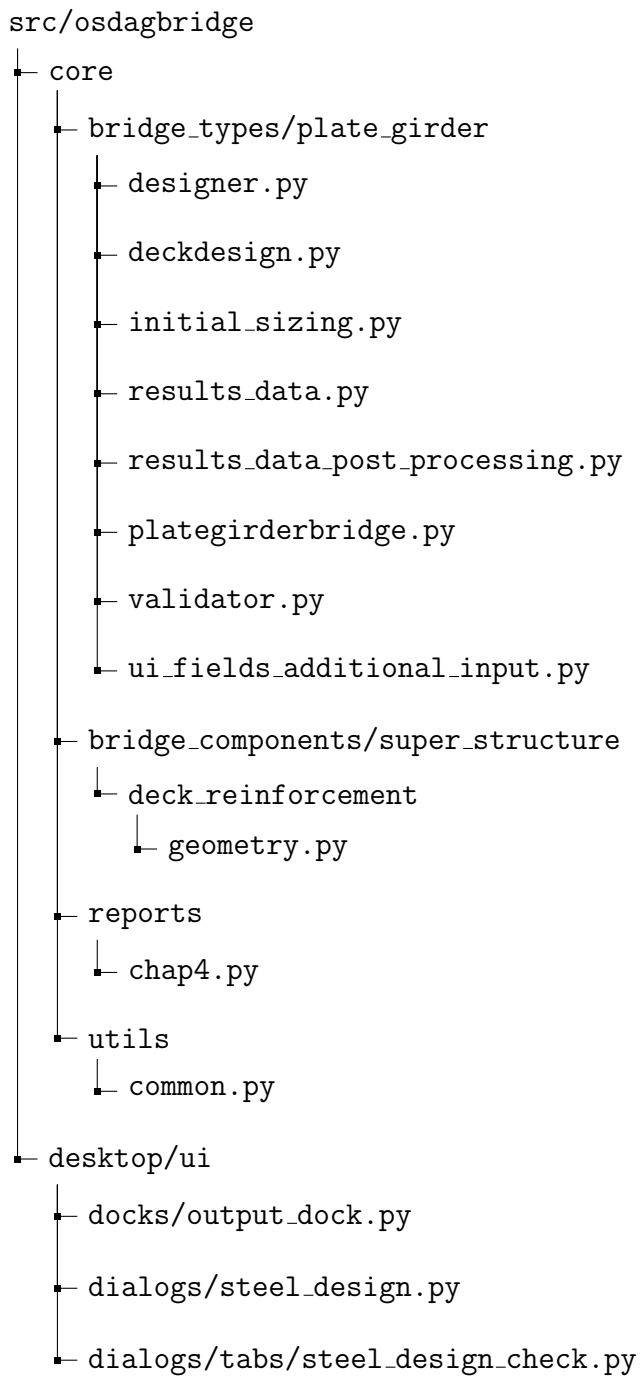
- **Listing 3.6**: the steel→composite correction is applied in exactly one place. Live deflection is the maximum over all vehicle positions, and all values leave under the `KEY_SD_DEFL_*_RAW` keys.

- **Listing 3.7–3.8**: camber is subtracted only from DL and DL+LL sags, clamped at zero and capped at 4 m. The cache keeps both raw and adjusted values; check #18 reports the residual DL sag, and checks 14/18 are gated on the limit so a zero result is still displayed.

- **Listing 3.9**: `run_design_check()` validates `result_data`, builds the camber-adjusted cache, writes it back onto the bridge object for Stage 7 (output dictionary / report) and hands it to the demand extractor.

3.4 Task 1: Documentation

3.4.1 Directory Structure



3.4.2 Module layout of designer.py

The designer is organised in six numbered sections:

1. **Bridge configuration** – input dataclasses `SteelProperties`, `SteelSection`, `SlabProperties`, `GeometryConfig`, `ShearStudConfig`, `FatigueConfig`, `StiffenerConfig` and the aggregate `BridgeConfig.from_plate_girder_bridge()`. All fields are required; missing inputs raise `ValueError`.

2. **Demand extractor** – `DemandEnvelope` (per girder and per load case, including `delta_dl_mm` and `camber_mm`).

3. **IRC 22:2015 capacity calculator** – `IRC22CapacityCalculator.compute_all()`; steel and composite-girder checks only.

4. **DCR engine** – `DCREngine` (checks 1–14, 18, stiffener 20–21) and `design_envelope_engine()`.

5. **Report generator** – `ReportGenerator` text summary.

6. **Main pipeline** – `run_design_check()`, `_extract_demands_from_analysis_results()`, `_compute_per_lc_dcr()`, `collect_girder_verdict()`.

3.4.3 Stage sequence

```
Stage 4G  analysis completes  ->  self.result_data =
   grillage_model.get_result_data()
Stage 5   run_design_check()  ->  build_deflections_cache (
   composite basis)
                                  ->
                                      apply_camber_to_deflections_cache
                                       (camber)
                                  -> DCREngine per girder / per load
                                       case
Stage 6   design_deck_slab()  ->  slab bars + Cl.604.4 / Cl
   .606.10 interface checks
                                      written back into
                                         design_results
Stage 7   output_dict         ->  KEY_SD_DEFL_* (post-camber) and
    KEY_SD_DEFL_*_RAW
```

```
                                    (pre-camber) per girder,
                                       consumed by reports
```

3.4.4 Keys added to common.py

- KEY_COMP_N, KEY_COMP_I, KEY_COMP_Y_TOP, KEY_COMP_Y_BOT, KEY_COMP_S_TOP , KEY_COMP_S_BOT, KEY_COMP_Y_FROM_BOT, KEY_COMP_AC_TRANS_MM2 – output of composite_section_properties().
- KEY_MP_GIRDER_CENTROID_YCG, KEY_MP_GIRDER_FLANGE_AREA_TOP, KEY_MP_GIRDER_FLANGE_ARE , KEY_MP_GIRDER_WEB_AREA – output of steel_i_section_properties().
- KEY_SD_DEFL_LIVE_RAW, KEY_SD_DEFL_TOTAL_RAW, KEY_SD_DEFL_DL_RAW – pre-camber deflections (report Chapter 4); KEY_SD_DEFL_AFTER_CAMBER, KEY_SD_APPLIED_CAMBER – post-camber values (report Chapter 5).

3.4.5 Developer notes

- Never recompute steel or composite section properties in a consumer; call steel_i_section_prope () / composite_section_properties() and read by key.
- Never add a literal default to a BridgeConfig field. Seed the value in the Additional Inputs defaults and read it with _req().
- Checks that need slab reinforcement belong in deckdesign.py, not in the steel DCR engine.
- The composite stiffness correction is applied only in results_data.py; camber is applied only in designer.py.

Chapter 4

Output Dock and Generate Results Integration

4.1 Task 2: Problem Statement

The Steel Design module of OsdagBridge had two disconnected result views. The Output Dock on the main window showed utilization (DCR) percent bars that were written once by the backend at the end of the design run, while the Steel Design dialog re-ran its own copy of the DCR pipeline to fill its check cards. The two views therefore computed the same quantities through different code paths and could disagree. Neither view responded to the Member / Load Case dropdowns, so the user could not inspect the DCR of a particular girder under a particular load combination.

The Steel Design Details tab and the Generate Results tables suffered from a related problem: their values were assembled from a merge of `input_dict`, the Additional Inputs working dictionary, the Input Dock values and the backend `output_dict`, with hard-coded fallbacks and defaults scattered through the UI layer. Several report tables (cross bracing, end diaphragm, seismic, wind, temperature, load combinations, bending/shear envelopes) were partly blank or showed stale/incorrect values because the required data was never persisted by the design pipeline.

The task was therefore to (i) make the backend the single source of truth for DCR computation and connect it to both the Output Dock and the Steel Design tab, driven by the Member and Load Case selectors, and (ii) make every value shown in the Steel Design Details tab and the Generate Results tables come only from the backend `output_dict`,

filling in the values that were missing.

4.2 Task 2: Tasks Done

Single DCR computation path (commit `38aa25c`)

- Added `PlateGirderBridge.get_dcr_engine_for_selection(girder, load_case)`, which rebuilds a `DemandEnvelope` from the stored per-girder / per-load-case demands, runs `IRC22CapacityCalculator` and `DCREngine`, and returns the engine. A thin wrapper `get_dcr_for_selection()` maps the check IDs to the eight utilization keys used by the percent bars.

- Removed the duplicated DCR pipeline from `steel_design.py`; the Design Check tab now calls the same backend method as the Output Dock.

- Output Dock: added `refresh_member_dropdown()`, `refresh_loadcase_dropdowns()` and `connect_design_dropdowns()` so the Member and Load Case combos are populated from the actual design (number of girders, load cases run) and any change re-fetches the DCR bars from the backend.

- Steel Design dialog: member/load combos mark the check results dirty and refresh immediately if the Design Check tab is visible.

- Refactored `designer.py`: `DCREngine` summary methods (`overall_status`, `max_dcr`, `n_pass` ...) now consider only the eight structural check categories, excluding stiffener checks; per-load-case demand extraction was extended with construction moment, self-weight moment, fatigue stress/shear ranges and the live-load shear range V_r.

- Moved all design-check metadata (check keys, order, titles, units, LaTeX and HTML equation strings, demand/capacity prefixes) into `core/utils/common.py` as shared constants. The Design Check tab was rewritten around these constants and now renders the equations as SVG through matplotlib mathtext with a cache.

Steel Design Details tab reads only `output_dict` (commits `2c9a276`, `73d31bc`)

- Removed all raw-input fallbacks from `steel_design_details.py`; every field (dimensional details, shear connector details, stiffener details) is read from the `KEY_SD_*` keys that `store_design_results()` writes.
- Stiffener inputs in the Additional Inputs dialog are now saved into the working dictionary on every field change so they reach the backend.
- Corrected section-property unit labels (cm → m) in the Additional Inputs UI fields.
- Made the tab girder-aware for the Custom design type: the backend now publishes each girder's own section (depth, flange/web dimensions, designation, class, effective slab width, restraints, web type) under `<key>.G{n}.M1`, and the Details tab reloads for the girder chosen in the selection bar.

Generate Results tables from `output_dict` (commits `f4916a5`, `2a0b94d`)

- Rewrote the cross-bracing and end-diaphragm resolvers to read the per-pair keys (`<base>.G1G2.B1M1` etc.), showing Yes/No for chord presence, human-readable section-type labels and the number of bracings.
- Refactored `generate_results_values_builder.py` (about 1000 lines changed) so that every `resolve_*` function takes only `output_dict`. The `GenerateResultsDialog` constructor was changed from `(input_dict, bridge)` to `(output_dict)` and the Output Dock no longer merges four dictionaries before opening it.
- Added report-only keys to `common.py` for permanent-load breakdown, camber/deflection, modular ratio and the full load-combination list.

Filling missing table values (commit `0311d47`)

- Seismic: zone factor Z now resolved from the selected seismic zone via IRC 6 Table 16; Z, S_a/g, A_h, A_v are stored in `output_dict`.

- Wind: basic wind speed, exposed height and terrain type are read from the proper input keys instead of hard-coded defaults.
- Temperature: effective bridge temperatures T_{min}, T_{max}, rise and fall computed through IRC 6 Cl. 215.2 and stored.
- Load combinations: the authoritative list (IRC 6 defaults + user selection + custom) is built once by `LoadCombinationWidget` and stored under `KEY_ALL_LOAD_COMBINATIONS`; the analyser names its ULS/SLS load cases from this list so the same string appears in the load-combination table, the force tables and the dropdowns.
- The load-effects envelope cache is persisted in `output_dict` so the bending-moment and shear-force envelope / by-load-case tables are no longer empty. Modular ratio is promoted to a flat key for the concrete material table.
- Generate Results page: the Member and Load Case dropdowns are populated from the number of girders and the load cases actually analysed. Added `available_load_cases()` and `filter_table()` so one filtered payload feeds the tree view, the preview and the Excel export.
- Cross-bracing defaults moved to a module-level `_CB_DEFAULTS` and applied by default; median load defaults to 0 kN/m instead of `None`.

4.3 Task 2: Documentation

Data flow

```
design()  -->  run_design_check()  -->  design_results["per_girder"][G]["per_lc"][LC]
                                   -->  store_design_results()  -->  output_dict

Output Dock  --(girder, load case)-->  backend.get_dcr_for_selection()
Steel Design --(girder, load case)-->  backend.get_dcr_engine_for_selection()
Details tab  --(girder index)------->  output_dict[KEY_SD_* + ".G{n}.M1"]
Generate Results  ----------------->  resolve_table(id, output_dict)
                                        -> filter_table(table, girder, load case)
```

Key files

- core/bridge_types/plate_girder/plategirderbridge.py — get_dcr_engine_for_selection, get_dcr_for_selection, store_design_results, _store_load_combinations_for_widget, _lc_name_by_key.
- core/bridge_types/plate_girder/designer.py — DCREngine._structural_checks, per-LC demand extraction.
- core/utils/common.py — KEY_OUTPUT_DOCK_*, KEY_CHECK_*, DESIGN_CHECK_ORDER/TITLES/UNIT KEY_ALL_LOAD_COMBINATIONS, KEY_SD_DL_*.
- desktop/ui/docks/output_dock.py — dropdown population and _on_design_selection_change
- desktop/ui/dialogs/steel_design.py, tabs/steel_design_check.py, tabs/steel_design_de
- desktop/ui/dialogs/generate_results_dialog.py, generate_results_values_builder.py — resolve_table, available_load_cases, filter_table.

Conventions introduced

- **DCR is computed in one place.** UI code never instantiates DCREngine; it asks the backend for a given (girder, load case).
- output_dict **is the only data source for result views.** Resolvers and tabs take output_dict alone; if a value is absent, the cell renders empty rather than falling back to an input default. Anything a table needs must be written by the backend at design time.
- **Per-girder keys** use the suffix .G{n}.M{m}; per-pair bracing/diaphragm keys use <base>.G{i}G{i+1}.<member>.
- **Load-case names** are generated once from the load-combination list and reused by the analyser, the caches, the force tables and every dropdown, so filtering matches by exact string.

Chapter 5

Conclusions

5.1 Tasks Accomplished

Over the course of the fellowship the following work was completed on the OsdagBridge Steel Design module:

- **Task 1:** [fill in from your Task 1 chapter]
- **Unified DCR computation.** The duplicated design-check pipeline in the Steel Design dialog was removed and replaced by a single backend entry point, `get_dcr_engine_for_sele` which both the Output Dock percent bars and the Steel Design check cards now call.
- **Interactive Output Dock.** The Member and Load Case dropdowns are populated from the actual design (number of girders, load cases analysed) and any change re-fetches the utilization bars from the backend, so a user can inspect the DCR of any girder under any load combination.
- **Design Check tab rewrite.** Check metadata (keys, order, titles, units, equations) was centralised in `common.py`; equations are rendered as cached SVG via matplotlib mathtext; the `DCREngine` summary now excludes stiffener checks from the eight structural categories.
- `output_dict` **as the single source of truth.** The Steel Design Details tab and all Generate Results resolvers were refactored to read only from the backend `output_dict`, removing hard-coded fallbacks and the four-dictionary merge that

previously fed the report dialog. The Details tab was made girder-aware for the Custom design type.

- **Completed report tables.** Seismic (Z, S_a/g, A_h, A_v), wind, temperature (IRC 6 Cl. 215.2), load combinations, bending/shear envelopes, modular ratio, cross-bracing and end-diaphragm tables that were blank or incorrect now carry values persisted at design time. Load-case names are generated once and reused across the analyser, force tables and dropdowns, and a common `filter_table()` feeds the tree view, preview and Excel export.

- **Bug fixes.** Unlimited zoom-out in the Bridge Plan and Cross Section views (Issue #211), cross-section hover behaviour, and the design-status logic.

5.2 Skills Developed

Technical

- **Python / PySide6 desktop development:** building and refactoring Qt widgets, docks, dialogs and tabs; signal/slot wiring; blocking signals during programmatic combo updates; custom SVG widgets.

- **Software architecture:** identifying duplicated computation paths and collapsing them into a single backend API; separating UI from core logic; designing a key-naming convention (`.G{n}.M{m}`, per-pair keys) for a flat result dictionary.

- **Structural engineering codes:** working knowledge of IRC 22:2015 and IRC 6:2017 clauses for flexure, shear, LTB, fatigue, deflection, seismic zone factors and effective bridge temperature, and how they map to demand/capacity ratios.

- **Data pipelines:** tracing values from user input through the grillage analyser and design checks into the report tables, and persisting intermediate results (load-effects cache, load-combination list) so downstream consumers do not recompute them.

- **Rendering and tooling:** matplotlib mathtext for LaTeX-to-SVG, Excel export, and working inside a conda environment with native dependencies.

- **Version control:** incremental, well-scoped commits on a feature branch; issue-linked fixes; reading and reviewing diffs.

Professional

- Reading and understanding a large, unfamiliar codebase (both OsdagBridge and upstream Osdag) before making changes.
- Debugging problems that span the UI, backend and analysis layers, and fixing root causes rather than adding fallbacks.
- Communicating design decisions (e.g. "output_dict only") in code comments and docstrings so future contributors follow the same convention.
- Working in an open-source, mentor-guided workflow: issues, branches, reviews and iterative feedback.

Chapter A

Appendix

A.1 Work Reports

OsdagBridge
Open Source Tools for Bridge Design

Summer Fellowship Work Report

— Daily Progress Log —

fossee
IIT BOMBAY
Free & Open Source Software for Education

Bridging Ideas for a Better Tomorrow

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

#	Date	Day	Task	Hours Worked
1	15-May-2026	Friday	Reviewed generate results schema and planned key mapping for input dock values	4
2	16-May-2026	Saturday	Studied designer.py flow and identified values needed for export tables	4
3	17-May-2026	Sunday	Tested Generate Results dialog preview/export with sample bridge inputs	3
4	18-May-2026	Monday	Started mapping design option keys to the generate results schema	5
5	19-May-2026	Tuesday	Continued key mapping for design options (cont.) section	5
6	20-May-2026	Wednesday	Debugged missing values in exported tables for design options	4
7	21-May-2026	Thursday	Cleaned up schema key naming and verified against input dock fields	4
8	22-May-2026	Friday	Mapped keys for design options and design options (cont.) to schema	6
9	23-May-2026	Saturday	Planned looped key generation for n-girder member properties	3
10	24-May-2026	Sunday	Implemented member properties key loop for multiple girders	5
11	25-May-2026	Monday	Added keys for member properties for n number of girders using loops	6
12	26-May-2026	Tuesday	Tested member property export for 2, 3 and 4 girder configurations	4
13	27-May-2026	Wednesday	Investigated equation rendering options for design check tab (SVG / PyMuPDF)	4
14	28-May-2026	Thursday	Prototyped SVG-based equation rendering in design check	5
15	29-May-2026	Friday	Implemented QPixmap text rendering of equations in design check; tested PyMuPDF & SVG approaches	6
16	30-May-2026	Saturday	Made Load Combination tables for ULS and SLS; mapped existing keys to show user-entered values in export	7
17	31-May-2026	Sunday	Improved the Load Combinations table UI	4
18	01-Jun-2026	Monday	Removed fallbacks from designer.py and verified design outputs	5
19	02-Jun-2026	Tuesday	Stored designer outputs in output dictionary; populated steel design details; added error handling for design function	7
20	03-Jun-2026	Wednesday	Connected Output Dock DCR to design check tab; fixed duplicate member property keys	6
21	04-Jun-2026	Thursday	Wired load case and member dropdowns to live DCR bars; removed duplicate keys from common.py	6
22	05-Jun-2026	Friday	Removed All/Envelope calculation from designer; included remaining forces in DCR calculations	7
23	06-Jun-2026	Saturday	Fixed design check percentage bug after merge; fixed values not showing in design details tab	5
24	07-Jun-2026	Sunday	Reviewed DemandEnvelope usage across steel design modules	3
25	08-Jun-2026	Monday	Changed DemandEnvelope to match designer.py; removed DemandExtractor from steel design file	6
26	09-Jun-2026	Tuesday	Started designer.py refactor; planned DCR connection for both tabs	5
27	10-Jun-2026	Wednesday	Refactored designer file and connected DCRs to both tabs; mapped keys for Loading tab in generate results	7
28	11-Jun-2026	Thursday	Generate results checkpoint – verified mapped tabs and fixed key mismatches	5

OsdagBridge
Open Source Tools for Bridge Design

Summer Fellowship Work Report

— Daily Progress Log —

fossee
IIT BOMBAY
Free & Open Source Software for Education

Bridging Knowledge for a Better Tomorrow

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

#	Date	Day	Task	Hours Worked
29	12-Jun-2026	Friday	Made tables for ULS checks, SLS checks, shear connector, stresses and deflection in generate results	7
30	13-Jun-2026	Saturday	Made all remaining tables for generate results	6
31	14-Jun-2026	Sunday	Made permanent/live load tables; fixed load combination tables; fixed errors after rebase	6
32	15-Jun-2026	Monday	Tested full generate results export in Excel and verified table structure	4
33	16-Jun-2026	Tuesday	Made the remaining two tables for generate results	5
34	17-Jun-2026	Wednesday	Fixed additional inputs editing after lock	4
35	18-Jun-2026	Thursday	End-to-end testing of generate results dialog with multiple bridge cases	4
36	19-Jun-2026	Friday	Reviewed steel design details tab code for migration to output dict	4
37	20-Jun-2026	Saturday	Documented output dictionary keys used by generate results tables	3
38	21-Jun-2026	Sunday	Reviewed generate results export output against design check tab values	3
39	22-Jun-2026	Monday	Investigated inconsistencies between output dock values and steel design details tab	4
40	23-Jun-2026	Tuesday	Refactored resolver helpers in generate_results_values_builder for reuse	5
41	24-Jun-2026	Wednesday	Cleaned up unused key lookups in ui_fields_additional_input	4
42	25-Jun-2026	Thursday	Tested additional inputs dialog after lock/unlock fixes	4
43	26-Jun-2026	Friday	Reviewed steel_design.py data flow into details and check tabs	4
44	27-Jun-2026	Saturday	Reviewed additional inputs and ui_fields_additional_input structure	3
45	28-Jun-2026	Sunday	Planned refactor of steel design details tab to read from output dict	4
46	29-Jun-2026	Monday	Refactored steel_design_details.py to remove hard-coded value lookups	5
47	30-Jun-2026	Tuesday	Values for steel design details tab now come from output dict	6
48	01-Jul-2026	Wednesday	Started moving CB/ED table values in results builder to output dict	5
49	02-Jul-2026	Thursday	CB ED table values from output dict in generate_results_values_builder	6
50	03-Jul-2026	Friday	Tested CB/ED tables against designer outputs and fixed mismatches	4
51	04-Jul-2026	Saturday	Reviewed remaining resolvers still reading from non-output-dict sources	4
52	05-Jul-2026	Sunday	Planned designer/deck design refactor and removal of design envelope case	3
53	06-Jul-2026	Monday	Started deck design refactor and initial sizing changes	5
54	07-Jul-2026	Tuesday	Continued designer refactor; prepared changes for review	5
			Total Hours Worked	**228**

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.