FOSSEE Winter Internship Report

On

Developing unit tests for Osdag using PyTest

Submitted by

Mayank Agarwal

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

Vellore Institute of Technology

Bhopal

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

February 5, 2026

Acknowledgments

- I would like to express my sincere gratitude to all those who supported and guided me throughout my internship with Osdag. This experience has been invaluable for my professional development in structural engineering software.
- My deepest appreciation goes to the entire Osdag team for their mentorship, particularly Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia for their patient guidance and technical expertise during my project work.
- I am honored to acknowledge the leadership of Prof. Siddhartha Ghosh, Principal Investigator of Osdag from the Department of Civil Engineering at IIT Bombay, for creating this impactful open-source initiative.
- Special thanks to Prof. Kannan M. Moudgalya, FOSSEE Project Investigator from the Department of Chemical Engineering at IIT Bombay, for his vision in supporting open-source engineering education.
- I gratefully acknowledge the support from FOSSEE managers Usha Viswanathan and Vineeta Parmar, along with their entire team, for creating opportunities that bridge academia and practical software development.
- This project was made possible through the support of the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India.
- I extend my thanks to VIT Bhopal University, particularly to Dr. Shriram R, Deputy Director - Placement and Training, for fostering an environment that encourages practical learning through such internship opportunities.

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.

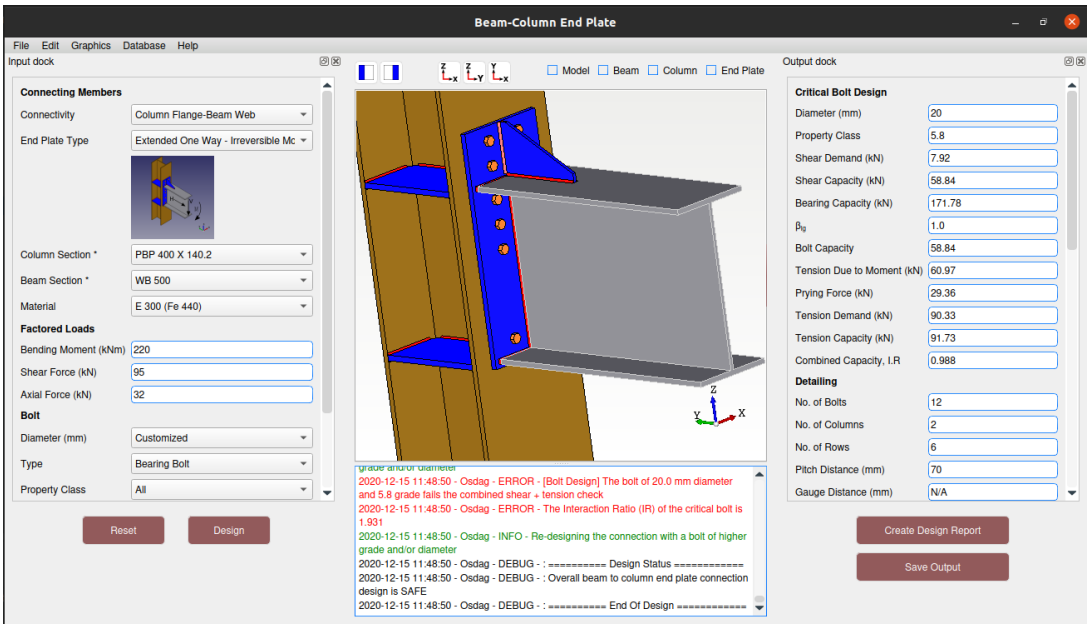

Figure 1.2: Osdag GUI

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

The screening task defined the selection criteria for the FOSSEE internship, requiring the development of a robust unit testing suite for the Osdag software. The primary objective was to ensure software reliability by verifying that the Osdag source code (specifically from the `migrate-to-click` branch) produced accurate design results when compared against a provided benchmark.

The specific problem requirements were:

- Objective: Develop automated unit tests using the Pytest framework to verify that the provided module code produces expected outputs.
- Target Modules: The task focused on validating multiple connection types, including Fin Plate Connections, Cleat Angle Connections, and Tension Members.
- Verification Method: The test suite had to execute Osdag designs using provided input files (`.osi`) and validate the resulting design capacities (e.g., Shear Capacity, Bolt Capacity, Tension Yielding) against a "Master Spreadsheet" of expected values.
- Constraints: The solution required a modular code structure, proper handling of relative paths, and adherence to Python PEP-8 coding standards.

2.2 Tasks Done

To successfully complete the screening task, the following technical activities were performed:

1. Environment Setup & Configuration

- Established a dedicated Anaconda virtual environment to manage dependencies such as `pythonocc-core`, `PyQt5`, and `sqlite3`.
- Configured the Pytest framework with a `conftest.py` file to manage test fixtures and relative paths, ensuring tests could run in any environment without hardcoded locations.

2. Test Infrastructure Development

- Developed a custom utility script, `test_helpers.py`, to programmatically trigger Osdag's design calculations. This script utilized `subprocess` and Osdag's CLI to run designs without launching the GUI.
- Implemented a result parsing engine to extract key design parameters (e.g., `Bolt.Capacity`, `Plate.Shear`) from Osdag's output dictionaries for comparison.

3. Unit Test Implementation

- Wrote distinct test modules for each required connection type:
  - `test_fin_plate_connection.py`: Validated Fin Plate design outputs against 4 distinct test cases.
  - `test_cleat_angle_connection.py`: Verified Cleat Angle behavior, specifically checking angle size selection and shear capacities.
  - `test_tension_member_welded.py`: Tested welded tension members for yielding and rupture failures.
- Used `assert_approximately_equal` logic to compare "Actual" vs. "Expected" values, allowing for minor floating-point tolerances.

4. Debugging & Optimization

- Resolved critical Circular Import Errors by restructuring imports and modifying `sys.path` in the test configuration.
- Fixed Database Connection Failures where the tests failed to locate `xsections.sqlite`, ensuring the headless test runner could access section properties correctly.
- Finalized the submission with a structured Git repository, including a comprehensive `README.md` and a clean `.gitignore` to exclude cache files.

Chapter 3

Internship Task 1: Report Generation Automation

3.1 Task 1: Problem Statement

The primary objective of this task was to design and implement an automated testing infrastructure to validate the Report Generation feature of the Osdag software. Following major refactoring efforts in the repository, manual verification of generated design reports became inefficient and unreliable.

The specific goals were:

- **Automation:** Trigger Osdag's report-generation pipeline programmatically without launching the full GUI.
- **Validation:** Ensure that valid design input files (`.osi`) result in correctly generated LaTeX (`.tex`) and PDF reports.
- **Quality Assurance:** Enforce size limits and verify LaTeX document structure to avoid corrupted or bloated output.
- **Integration:** Integrate the test suite into the existing `pytest` framework for long-term maintainability and CI usage.

3.2 Task 1: Tasks Done

A comprehensive automated testing workflow was implemented using Python and `pytest`. The workflow closely mirrors a real user invoking Osdag from the command line.

Testing Workflow

1. **Environment Setup:**

   Heavy graphical dependencies such as OCC were mocked to allow headless execution.

2. **Database Initialization:**

   The engineering database (`Intg_osdag.sqlite`) is reset and validated before every test run.

3. **CLI Execution:**

   The Osdag CLI is invoked using:

   `python -m osdag_gui cli run -i <input.osi> -t generate_report`—

4. **Validation:**

   The generated LaTeX file is inspected for:

   i. Existence ii. Non-zero size iii. Maximum size threshold

5. **Cleanup:**

   Output directories are cleared after each run.

3.3 Task 1: Python Code

This section presents the Python automation script developed during the internship for validating Osdag's report generation pipeline. The script executes the CLI, triggers report creation, and verifies the generated outputs against defined quality criteria.

3.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters:** Accepts `.osi` design files either from the default test suite or via command-line arguments.
- **CLI Automation:** Executes Osdag in headless mode using Python's `subprocess` module.
- **Validation Logic:** Searches for generated `.tex` files, verifies their size, and inspects LaTeX headers.
- **Cleanup:** Removes generated files after execution.

3.4 Acceptance Criteria

For a test to be marked as successful, the generated report was required to satisfy all of the following conditions:

- The LaTeX file must exist on disk.
- The file size must be greater than zero.
- The file size must remain below a defined threshold to prevent bloated output.
- The document must contain a valid LaTeX document class declaration.

3.5 Task 1: Documentation and Usage

The testing infrastructure was documented so that future contributors can easily execute and extend the test suite.

Default Execution

Running the automated test suite executes the predefined representative design cases for multiple Osdag modules.

Custom Input Testing

The framework allows developers to specify custom design input files at runtime, enabling rapid verification of newly added or modified models.

Reporting

At the end of every execution, the test output summarizes the size and status of the generated reports for each input case.

3.6 Current Status

At the conclusion of the internship period, the report automation framework was fully functional and validated across multiple Osdag design modules. The average execution time per report was within a few seconds, enabling efficient regression testing after future code changes.

Chapter 4

Internship Task 2: CLI Debugging and Refactoring Support

4.1 Task 2: Problem Statement

During the internship period, the Osdag repository underwent significant structural refactoring that affected several subsystems, particularly the Command Line Interface and the internal database initialization process. These changes introduced failures when the software was executed in non graphical environments such as automated test pipelines.

Common issues observed included missing database tables, incorrect path resolution for engineering data files, and changes in the invocation pattern of the CLI after the migration to a new command framework. These regressions prevented automated report generation and blocked the execution of the newly developed testing framework.

The objective of this task was therefore to analyze the failures introduced by the refactoring, validate the new CLI workflows, identify root causes for database related errors, and assist the core development team by documenting reproducible failure cases and verified execution procedures.

4.2 Task 2: Tasks Done

To support the refactoring effort and restore reliable CLI based execution, the following technical activities were carried out.

Branch Synchronization and Analysis

Latest changes were regularly fetched from upstream repositories and tested across multiple branches in order to identify behavioral differences between development versions. Controlled merges were performed to integrate report testing features with updated CLI implementations while avoiding regressions in the main codebase.

CLI Workflow Verification

The updated invocation pattern for executing Osdag from the terminal was examined and validated. Various execution modes were tested to confirm that report generation could be triggered without launching the graphical interface. Special attention was paid to argument parsing, output directory handling, and default configuration paths.

Database Debugging

Failures related to missing tables inside the engineering database were investigated. The database initialization logic was traced to determine where schema creation was skipped in certain execution paths. Environment specific path issues were documented, especially on Windows systems, where relative paths resulted in incorrect database resolution.

Failure Reproduction and Root Cause Analysis

Observed runtime errors were systematically reproduced in isolated environments. Stack traces were captured and correlated with recent commits to locate the source of the failures. Particular focus was placed on validation routines within connection design modules that assumed unavailable input values during headless execution.

Cross Branch Testing

The compatibility of new test infrastructure with refactored CLI workflows was evaluated by executing the same test cases across multiple branches. This ensured that testing improvements did not conflict with ongoing architectural changes in the core application.

Collaboration and Communication

Findings were communicated to maintainers through technical summaries, discussion threads, and documented reproduction steps. Proposed fixes and configuration updates were shared for review, enabling faster resolution of integration issues.

4.3 Task 2: Documentation

Detailed technical documentation was maintained throughout the debugging process in order to assist both maintainers and future contributors.

Branch Strategy

A clear branching strategy was documented for development and testing workflows. Feature branches were used for isolated experimentation, while upstream test branches were periodically merged to remain synchronized with the latest refactoring changes.

Environment Setup Guide

Step by step instructions were prepared for configuring the Conda environment, installing required dependencies, and setting system variables required for CLI execution on different platforms.

Troubleshooting Reference

A troubleshooting guide was compiled listing common runtime errors, their observable symptoms, and the verified steps required to resolve them. This included guidance for database initialization failures, missing resource files, and incorrect CLI invocation patterns.

Reproducibility Notes

All experiments were logged with branch names, commit identifiers, and operating system details to ensure that reported issues could be reliably reproduced by other developers.

Chapter 5

Conclusions

5.1 Tasks Accomplished

During the FOSSEE Winter Internship, substantial contributions were made to the Osdag open–source project with the objective of enhancing its reliability, testability, and command–line usability. The major accomplishments are summarized below:

- **Automated Testing Framework Development**
  A comprehensive automated testing framework was designed and implemented to verify the complete report–generation pipeline of Osdag. This framework enabled headless execution of structural design workflows through the Command Line Interface (CLI) and ensured correct generation of LaTeX and PDF reports across multiple design modules.

- **Validation of Refactored CLI**
  Systematic testing was carried out across several repository branches to validate the refactored CLI. Execution failures introduced during architectural changes were identified, reproduced, and thoroughly documented. Special attention was given to database initialization errors, incorrect path resolution, and missing schema definitions that hindered successful design validation and report creation.

- **Cross–Branch Compatibility Testing**
  Extensive cross–branch testing ensured that the newly introduced testing infrastructure remained compatible with ongoing core refactoring efforts. Controlled merges

were performed to synchronize feature branches with upstream development while preserving stability within the testing environment.

- **Technical Documentation and Knowledge Transfer**
  Detailed technical documentation was produced covering automated testing workflows, branch synchronization strategies, CLI execution patterns, environment setup procedures, and troubleshooting guidelines. This documentation enables future contributors to easily reproduce results and extend the testing framework with minimal overhead.

- **Collaboration and Communication**
  Continuous collaboration was maintained with core maintainers and fellow interns throughout the internship. Findings were communicated through structured discussions, technical summaries, and written reports, contributing to faster debugging cycles and improved coordination within the distributed development team.

5.2 Skills Developed

The fellowship provided a valuable opportunity to strengthen both technical and professional competencies through hands–on involvement in a large–scale open–source engineering application.

- **Test Automation Expertise**
  Advanced proficiency was developed in Python–based test automation using the *pytest* framework, including fixture design, dynamic test parametrization, subprocess–driven CLI execution, and structured validation of file–based outputs.

- **Debugging and System Analysis**
  Strong debugging skills were cultivated through the investigation of complex runtime failures across multiple modules, analysis of stack traces, and identification of root causes within a rapidly evolving codebase. Significant experience was gained in diagnosing database–driven application issues, particularly schema initialization failures and platform–specific path resolution problems.

- **Software Engineering Practices**
  Practical exposure was obtained in Git–based version control, branching strategies,

upstream synchronization, and conflict resolution. Regular interaction with maintainers further strengthened understanding of continuous integration pipelines and collaborative development workflows.

- **Professional and Communication Skills**

  The internship enhanced technical communication abilities through the preparation of structured documentation, detailed bug reports, and reproducible case studies. Participation in an open–source community fostered discipline in progress reporting, adherence to coding standards, and coordination across geographically distributed teams.

- **Overall Impact**

  Collectively, the internship established a strong foundation in applied software testing, debugging, and collaborative development within a research–driven open–source environment.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report	
Name:	Mayank Agarwal
Project:	Osdag
Internship:	FOSSEE Winter Internship 2025–26

DATE	DAY	TASK	Hours
01-Dec-2025	Monday	Installed Osdag repository on Windows, created conda environment, and verified Python and dependency versions.	4
02-Dec-2025	Tuesday	Configured Osdag editable install (pip install -e .) and resolved initial package dependency issues.	4
03-Dec-2025	Wednesday	Explored Osdag folder structure (src, osdag_core, osdag_gui, tests) and studied README / developer docs.	4
04-Dec-2025	Thursday	Tested basic CLI and GUI launch commands to confirm Osdag runs correctly on the local machine.	4
05-Dec-2025	Friday	Set up pytest in the environment, checked existing test configuration (pytest.ini, conftest.py).	4
06-Dec-2025	Saturday	Cloned required branches (master, test, report-tests) and practiced switching/merging locally.	4
07-Dec-2025	Sunday	Reviewed FOSSEE screening task files and earlier unit tests to understand expected coding style.	4
08-Dec-2025	Monday	Documented installation steps and environment setup procedure for reproducibility on other systems.	4
09-Dec-2025	Tuesday	Verified database files and resources used by Osdag (SQLite, section properties, images).	4
10-Dec-2025	Wednesday	Ran sample Osdag designs manually using .osi input files to confirm design calculations run end-to-end.	4
11-Dec-2025	Thursday	Validated that Osdag generates design results correctly for fin plate and tension member examples (without reports).	4
12-Dec-2025	Friday	Installed LaTeX tools and checked basic .tex compilation flow used by Osdag report generator.	4
13-Dec-2025	Saturday	Reviewed report generation modules (reportGenerator_latex, CreateLatex) and output folder structure.	4
14-Dec-2025	Sunday	Mapped out high-level plan for report testing: .osi $\rightarrow$ CLI/design $\rightarrow$ LaTeX report $\rightarrow$ file validation.	4
15-Dec-2025	Monday	Finalized setup checklist (environment, database, LaTeX, tests) and confirmed everything works consistently.	4
16-Dec-2025	Tuesday	Designed initial pytest structure for report generation testing and created test_report_generation.py skeleton.	5
17-Dec-2025	Wednesday	Implemented helper utilities to locate .osi files and prepare output directories for generated reports.	5
18-Dec-2025	Thursday	Wrote first end-to-end test case for FinPlate .osi file, focusing on existence of generated .tex report.	5
19-Dec-2025	Friday	Enhanced tests to check report size limits and basic LaTeX structure in the generated .tex files.	5
20-Dec-2025	Saturday	Refactored helper code to avoid circular imports and cleaned up sys.path usage in tests.	5
21-Dec-2025	Sunday	Analyzed failures related to missing metadata (reportsummary) and studied how Osdag passes data to CreateLatex.	5
22-Dec-2025	Monday	Discussed with mentor about preferred approach: direct Python API vs CLI-based execution for report testing.	5
23-Dec-2025	Tuesday	Started integrating Osdag CLI into tests using subprocess instead of direct imports, prepared initial command patterns.	5
24-Dec-2025	Wednesday	Validated CLI design run for sample .osi files and captured logs to understand full execution flow.	5
25-Dec-2025	Thursday	Observed SQLite errors (no such table) during CLI execution and began tracing database initialization logic.	5
26-Dec-2025	Friday	Checked database configuration scripts and verified locations of section property tables used by Osdag.	5
27-Dec-2025	Saturday	Created a dedicated database setup helper (setup_db script) and integrated it into the pytest workflow.	5
28-Dec-2025	Sunday	Re-ran CLI-based tests with proper database setup and confirmed design calculations complete successfully.	5

DATE	DAY	TASK	Hours
29-Dec-2025	Monday	Implemented final CLI-based report test: .osi $\rightarrow$ python -m osdag_gui cli run -t generate_report -o output.	6
30-Dec-2025	Tuesday	Added search logic to locate the expected .tex files in the output directory and handle different naming patterns.	6
31-Dec-2025	Wednesday	Introduced command line option –osi for pytest to allow testing of custom .osi files without code changes.	6
01-Jan-2026	Thursday	Extended test coverage to additional modules (TensionWelded, CleatAngle) using the same report validation logic.	6
02-Jan-2026	Friday	Fine-tuned acceptance criteria: non-empty file, size $<$ 100 KB, and presence of \documentclass{report} or article.	6
03-Jan-2026	Saturday	Measured test runtimes $\sim$4–6 seconds per report and optimized logging to keep pytest output readable.	5
04-Jan-2026	Sunday	Documented the full report testing methodology, workflow steps, and acceptance criteria for internal use.	6
05-Jan-2026	Monday	Reviewed branch differences after upstream refactoring and adjusted tests to remain compatible with new CLI structure.	6
06-Jan-2026	Tuesday	Helped debug additional CLI failures (module validation errors) and shared reproducible steps with maintainers.	6
07-Jan-2026	Wednesday	Polished test code style, added comments, and improved error messages for easier debugging by other contributors.	6
08-Jan-2026	Thursday	Prepared concise documentation for developers explaining how to run and extend the report generation test suite.	6
09-Jan-2026	Friday	Verified that the complete test pipeline runs reliably on fresh environments and different branches.	6
10-Jan-2026	Saturday	Summarized key findings and open issues related to report generation and CLI/database behavior.	6
11-Jan-2026	Sunday	Drafted sections for the main internship report describing Task 1 (report automation) and Task 2 (CLI debugging).	6
12-Jan-2026	Monday	Reviewed and refined documentation with mentor feedback, corrected wording and added missing technical details.	6
13-Jan-2026	Tuesday	Did a final pass on code, ensuring repository cleanliness, ignoring caches, and organizing test directories.	6
14-Jan-2026	Wednesday	Completed final internship report draft, including chapters on screening task, internship tasks, and conclusions.	6
15-Jan-2026	Thursday	Prepared final deliverables (code, documentation, work report) and shared with mentor for closing review.	6
16-Jan-2026	Friday	Addressed minor review comments and ensured all links, paths, and commands in the documentation are correct.	5
17-Jan-2026	Saturday	Double-checked test executions and recorded screenshots/logs for future reference in the project.	5
18-Jan-2026	Sunday	Organized all internship artifacts (tests, docs, reports) into a structured folder for archival.	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.