Addis Ababa Science and Technology University

College Of Electrical and Mechanical Engineering

Department Of Software Engineering

Open Source Software Paradigms

Project Report

Group - 5

Group members **Id**

1. Dawit Sema…….....ETS 0455/14

2. Dawit Birhanu…….....ETS 0457/14

3. Delal Mohammad …….....ETS 0482/14

4. Eden Melaku…….……...….... ETS 0496/14

5. Eyerusalem Kidane…… ETS 0574/14

6. Gelila Nebiyu…….....……...ETS 0690/14

7. Hana Abiyu ……ETS0728/14

Submitted to: Mr. Biniyam B

Submitted Date – December 25, 2025

Flutter Clean Architecture VS Code Extension (Extended) Report

1. Introduction and Project Goals

The “Flutter Clean Architecture VS Code Extension (Extended)” project is an open-source Visual Studio Code extension designed to automate feature scaffolding for Flutter applications that follow Clean Architecture principles. The project builds on an existing extension that originally supported only the BLoC pattern and extends it to include Riverpod and Provider state management options while maintaining a consistent architectural structure.

The primary goals of the project are:

- To provide an open-source tool that generates Clean Architecture feature boilerplate for Flutter projects directly from VS Code.

- To extend the original BLoC-based generator with additional support for Riverpod and Provider, giving developers flexibility in their choice of state management.

- To publish the project on a public code hosting platform with appropriate licensing and comprehensive documentation (including README, license, contribution guidelines, and code of conduct).

- To promote the project within relevant developer communities and collect feedback from potential users and contributors.

2. Development Process

2.1 Phase 1 – Research and Setup

The work began with a detailed review of the original Flutter Clean Architecture starter template and its associated VS Code extension. The objective was to understand the existing folder structure (core, data, domain, presentation), the generation commands, and how the BLoC pattern was integrated into the generated code. In parallel, the licensing of the original project was examined to ensure that the chosen license for the extended work (MIT) was compatible with the upstream repository.

2.2 Phase 2 – Extension Implementation

In the implementation phase, the existing BLoC feature creation flow was preserved to maintain backward compatibility for users who already rely on the original behavior. On top of this, a pattern selection step was introduced using a quick pick interface in VS Code, allowing the user to choose BLoC, Riverpod, or Provider when generating a new feature.

For Riverpod, new templates were created to generate provider definitions, screens implemented with ConsumerWidget, and integration points for invoking use cases from the domain layer. For Provider, the project added templates that generate ChangeNotifier classes, provider wiring, and screens built using Consumer widgets. All three patterns share the same underlying Clean Architecture structure; only the presentation-layer files differ depending on the selected state management approach.

2.3 Phase 3 – Testing and Packaging

Once the generation logic for all three patterns was in place, the extension was tested by creating features with different names (for example, profile and product) and verifying that the folder structure and imports matched expectations. Particular attention was paid to ensuring that the generated code compiled, that the architecture remained consistent across patterns, and that the commands behaved correctly in typical Flutter projects.

After validating the behavior, a tagged release (version 1.1.0) was created to mark the addition of Riverpod and Provider support. This release serves as a clear milestone for the extended functionality and allows users to reference a specific version that includes the new patterns.

2.4 Phase 4 – Documentation and Licensing

The final development phase focused on documentation and formal licensing. The README.md file was expanded to include an overview of the extension, a description of its features, installation instructions, usage examples, and separate subsections describing how BLoC, Riverpod, and Provider are supported. Example folder structures were added so that users can quickly understand how generated features are organized.

In addition, the project repository was supplemented with LICENSE.md (using the MIT license), CONTRIBUTING.md to describe how new contributors can participate, and CODE_OF_CONDUCT.md to establish expectations for behavior within the community. These documents align the project with common open-source best practices and make it easier for others to collaborate.

3. Challenges and Solutions

3.1 Consistent Architecture Across Patterns

A key challenge was maintaining a consistent Clean Architecture layout while introducing multiple state management patterns. The solution was to treat the original template's architecture core, data, domain, and presentation as the canonical structure, and restrict pattern-specific variability to the presentation layer. In practice, this meant that entities, repositories, use cases, and data sources are generated identically regardless of pattern, while only the UI and state-management wiring differ.

3.2 Idiomatic Templates for Each State Management Tool

Designing templates that felt natural and idiomatic for each state management library was another challenge. This required revisiting official documentation and community examples for Riverpod and Provider to understand common patterns of use. For Riverpod, the templates adopted patterns such as StateNotifier and StateNotifierProvider for state mutations and exposure. For Provider, the templates relied on ChangeNotifier classes combined with ChangeNotifierProvider and Consumer to update the UI in response to state changes. This approach ensures that generated code is not only functional but also recognizable to developers already familiar with these tools.

3.3 Documentation for Less Experienced Users

Because the extension targets students and junior developers, clarity in documentation was essential. Early drafts of the README assumed substantial prior knowledge and were refined to provide more explicit installation steps, clearer usage instructions, and pattern-specific examples. The final documentation aims to guide a user who has only basic experience with Flutter and VS Code from installation through to generating and using a new feature.

4. Community Engagement Strategy

4.1 Promotion and Outreach

To position the project as a useful resource for the wider community, a simple engagement strategy was defined. The repository and release information were shared in campus programming groups and Flutter-focused communities, including channels where developers discuss tooling and project architecture. The messaging emphasized the problem of repetitive manual creation of folders and boilerplate in Clean Architecture projects and the solution: a single command that generates the necessary structure while allowing the user to choose their preferred state management pattern.

4.2 Onboarding Contributors

The project was prepared for external contributions by keeping the main README concise and pointing contributors to the dedicated CONTRIBUTING.md and CODE_OF_CONDUCT.md documents. Within the issue tracker, it is planned to label suitable tasks as “good first issue” and “help wanted”, which is a common technique to help newcomers identify approachable entry points in open-source projects. This structure lowers the barrier for students and early-career developers who wish to practice contributing.

4.3 Feedback and Iteration

Users are encouraged to provide feedback via GitHub issues, particularly for missing patterns, enhancements to the generated templates, and any bugs in folder creation or imports. Planned future enhancements include better integration with popular routing solutions, additional configuration options for folder naming, and more advanced examples. This feedback loop ensures that the extension can evolve according to real-world needs rather than remaining a static academic exercise.

5. Branding and Visual Identity

The branding for the project is intentionally simple and aligned with its purpose. The name “Flutter Clean Architecture VS Code Extension (Extended)” clearly indicates both the technology (Flutter), the architectural focus (Clean Architecture), the platform (VS Code), and the fact that this project extends an existing tool with additional capabilities. The original Flutter logo is retained as part of the visual identity to make it immediately obvious that the tool is intended for Flutter projects.

Consistency is maintained across the GitHub repository, any potential Marketplace listing, and supporting presentation materials. The same project name, a short descriptive tagline, and a brief explanation of the added Riverpod and Provider support are reused so that users encounter a coherent identity regardless of where they discover the project.

6. Overall Impact and Feedback

The extension meaningfully reduces the time and effort required to set up new features in Flutter projects that apply Clean Architecture. Instead of repeatedly creating directories, boilerplate classes, and state management wiring by hand, developers can invoke a single command, enter a feature name, choose a pattern, and receive a complete, structured starting point. This accelerates onboarding for new team members and allows students to spend more time on core application logic and less on repetitive setup.

By supporting BLoC, Riverpod, and Provider within the same tool, the project also serves as an educational resource. It allows learners to compare how different state management patterns are wired into an identical underlying architecture and to experiment with them in parallel. Early informal feedback from peers has been positive, especially regarding the convenience of having multiple patterns in one extension. Suggestions for improvement have highlighted the value of additional examples, short videos demonstrating each pattern, and more advanced scenarios, which form a clear roadmap for future work.

Overall, the project achieves the objectives set for the open-source software development course: creation of a functional open-source tool, adherence to licensing and documentation standards, meaningful technical enhancement of an existing codebase, and initial engagement with a developer audience.