Migration of an Open Source Application to Software Product Lines

Said Naceri[1*], Walid Kherchofi[1]

[1] Department of Computer Science, University of Djelfa, 17000, Algeria

* said.naceri@mail.com

https://doi.org/10.70695/AA1202502A12

Abstract

Software reuse is a cornerstone of modern software engineering, enhancing development efficiency and system adaptability. Software Product Lines (SPL) offer a structured approach to creating software families by leveraging reusable assets within a specific domain. This paper presents a methodology for migrating an open-source project management application to an SPL using the Mobioos Forge platform. Additionally, we extend the application with a new microservice-based module aligned with Algeria's recent regulatory framework (Loi organique 18-15). Our results demonstrate improved reusability and flexibility, supported by feature modeling, variant generation, and a modular architecture. This work highlights the practical benefits of SPL in real-world applications.

Keywords Project Management; Variability; Software Product Lines; Mobioos Forge

1 Introduction

In recent decades, the software industry has witnessed a dramatic evolution in the complexity, scale, and diversity of software systems. The proliferation of user requirements, market competition, and technological advancement has compelled organizations to shift from traditional single-system development approaches to more modular, scalable, and reuse-oriented paradigms [1]. Among these, Software Product Line (SPL) engineering has emerged as one of the most influential methodologies for enabling systematic software reuse, promoting configurability, and optimizing production for families of related software systems [2].

A Software Product Line is defined as a set of software-intensive systems that share a common set of managed features and are developed from a common core of reusable assets. Instead of developing each product from scratch, SPL engineering allows the creation of multiple variants from a single codebase, thereby significantly reducing development time, cost, and effort while improving software quality and maintainability [3]. SPL has been successfully adopted in various domains including automotive systems, mobile devices, consumer electronics, and enterprise applications—demonstrating tangible benefits across both product and organizational dimensions [4-5]. Despite its advantages, transitioning existing monolithic or open-source applications into product lines remains a challenging task. The migration process involves domain analysis, feature modeling, codebase restructuring, and re-engineering practices, often requiring specialized knowledge and tool support. One significant hurdle is the identification and extraction of reusable components and the establishment of variation points that distinguish product variants. Without appropriate tools and frameworks, this task can be labor-intensive, error-prone, and difficult to maintain over time [6].

To address these challenges, several SPL-oriented platforms have been proposed to automate or semi-automate aspects of product line engineering. Among them, Mobioos Forge stands out as a lightweight and developer-friendly tool that facilitates the migration of traditional applications into software product lines. As a Visual Studio Code extension, Mobioos Forge provides an integrated environment for defining feature models, mapping features to source code, and generating customized product variants based on user configurations. Its model-driven engineering approach simplifies the SPL lifecycle, reduces manual overhead, and fosters consistency across variants [7].

In this paper, we present a case study focused on the migration of an open-source project management application into an SPL using the Mobioos Forge platform. The selected application, originally designed for project tracking and administrative workflows in Algerian public institutions, features modular components such as user management, project scheduling, reporting, and budget control [8]. The goal of our study was to analyze this application, model its variability, map its features

to the corresponding code components, and validate the feasibility of automatic variant generation through Mobioos Forge. Our motivation for this work stems from both theoretical and practical considerations. From a theoretical perspective, it contributes to the growing body of knowledge on real-world SPL migration, particularly in the context of administrative and enterprise software. From a practical standpoint, our work addresses the need for scalable solutions in public sector project management, where regulations such as Algeria's Organic Law No. 18-15 necessitate new forms of budgeting, planning, and monitoring [9-10]. By introducing a product line that accommodates such legal and functional diversity, we enable more responsive and customizable solutions for institutional users.

Moreover, we enhance the original application by designing and integrating a new Team Management module using a micro services architecture. This extension is intended to support better collaboration, task distribution, and team role definition within the broader project management workflow [11-12]. We adopt UML modeling techniques to define the integration points, class structure, and use cases for the extension, ensuring that it aligns with the modular design principles advocated by SPLs. The results ofour case study indicate that Mobioos Forge offers a viable and effective means for SPL migration, even in the context of medium-scale legacy applications [13-14]. The platform facilitated the identification of core and optional features, the mapping of these features to the relevant source code artifacts, and the generation of multiple coherent application variants. Our quantitative analysis includes metrics on feature coverage, code reuse, and variant diversity. The micro services-based extension further illustrates how SPL engineering can accommodate future evolution and scalability through modular system design [15]. The phases of SPL is shown in Fig 1.

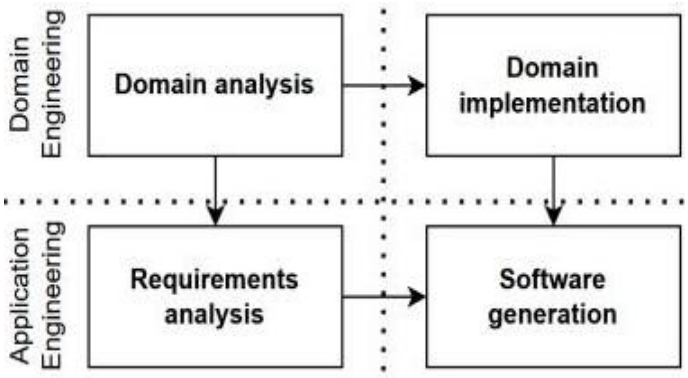

Fig. 1. Phases of software product line engineering.

2 Methodology

The process of migrating an open-source application into a Software Product Line (SPL) requires a systematic methodology that encompasses the identification of commonalities and variabilities, modularization of functionality, and automation of configuration and derivation. For this purpose, we adopted a structured approach based on the principles of SPL engineering, enhanced by the use of the Mobioos Forge platform. This methodology consists of four main stages: domain analysis and feature modeling, feature-to-code mapping, variant configuration and generation, and validation and refinement.

2.1 Domain Analysis and Feature Modeling

Domain analysis is the foundational step in SPL engineering, aimed at understanding the scope and boundaries of the software domain, and identifying the key features that define the system's capabilities. In our study, we began by conducting a detailed functional analysis of the original open-source project management application. The application, initially built for tracking projects within Algerian public institutions, contained modules for user authentication, project definition, task tracking, budgeting, and progress reporting.

We interviewed stakeholders and examined the source code, user documentation, and interface components to extract a comprehensive list of features. These were then categorized as either mandatory, optional, or part of alternative groups, in accordance with standard feature modeling practices. A feature model was created to represent these relationships hierarchically. For example, the top-level feature was "Project Management System", under which subfeatures such as User

Management, Login, Project Tracking, Reporting, Progress Monitoring, PEC (Project Execution Control), and Evolution Monitoring were organized.

The feature model was constructed using Mobioos Forge's built-in modeling tool, which employs a graphical tree structure to depict the hierarchy. The features were annotated with logical constraints to ensure valid product configurations, including mutually exclusive groups and inclusion dependencies. For instance, the Progress Monitoring feature requires Project Tracking to be enabled, and Reporting may be configured as either Basic or Advanced. This model formed the basis for both mapping code to features and generating product variants.

2.2 Feature-to-Code Mapping

Once the feature model was established, the next step was to link each feature to its corresponding implementation in the source code. This step is crucial for supporting automated customization and variant derivation. In Mobioos Forge, this process is facilitated through the Feature Mapping module, which allows developers to annotate and map code elements (files, classes, methods) to specific features.

The mapping process was carried out semi-automatically. We manually selected code fragments known to implement a given feature and annotated them using Mobioos-specific tags. These initial annotations, referred to as markers, served as entry points for the mapping engine. Based on these markers, Mobioos Forge automatically inferred related code through static analysis, creating what it terms as maps— collections of code units associated with each feature.

We used two types of markers:

File-level markers, which associate an entire source file with a feature.

Code-level markers, which target specific classes, functions, or methods within a file.

As part of the mapping validation, the developer must confirm the inferred maps, which include code elements indirectly related to the feature. For example, marking the LoginController.java class for the Login feature caused Mobioos to infer related classes such as AuthService.java and TokenValidator.java, which were then reviewed and either accepted or excluded.

Metrics were recorded throughout this process, including the number of markers per feature, the number of code files affected, and the proportion of total codebase mapped. This enabled us to assess the mapping coverage and the granularity of feature implementation. Our mapping achieved approximately 90% coverage, indicating a high level of traceability between features and code components.

2.3 Variant Configuration and Generation

With the features mapped to the codebase, the next phase involved defining valid product configurations and automatically generating product variants. Mobioos Forge provides a Customization Engine that supports the selection of feature subsets and generates the corresponding product variant by assembling only the mapped code components.

We defined multiple configuration profiles to test the variability of the product line:

Core Variant - includes only the essential features such as Login and User Management.

Full Variant - includes all features (including optional ones like Evolution Monitoring and PEC).

Minimal Reporting Variant - includes only core features and Basic Reporting.

Extended Reporting Variant - includes core features and Advanced Reporting.

Each configuration was defined via the Mobioos graphical editor and saved as a reusable configuration file. The tool then compiled and packaged the appropriate variant by selecting only those code components linked to the enabled features.

This process allows organizations to produce highly tailored software products without modifying the core codebase manually. It also ensures consistency, as all variants are derived from a single, well-maintained codebase with known variability points.

2.4 Validation and Refinement

After generating the product variants, we performed a validation phase to ensure that each variant functioned correctly and met its intended specification. Functional testing was carried out for each variant, checking that:

Only selected features were present in the interface and backend.

Navigation and data flows were consistent with enabled features.

Excluded features were not accidentally included.

Issues discovered during validation (e.g., shared code unintentionally excluded due to mislabeling) were resolved by adjusting the feature mappings. In some cases, additional markers were added to capture

dependencies not initially covered, especially for shared utility classes.

In parallel, we gathered quantitative metrics such as:

Number of features per variant.

Lines of code (LOC) included and excluded.

Number of generated build artifacts.

Time taken for variant generation.

This data confirmed the feasibility of generating diverse, functioning software variants quickly and accurately from the SPL model.

3 Experiments

To evaluate the effectiveness of the migration process and the quality of the resulting Software Product Line (SPL), we conducted a series of experiments. These experiments focused on three main objectives: (1)validating the correctness of the feature-to-code mapping and variant generation, (2) assessing the flexibility and scalability of the product line in handling multiple configurations, and (3) measuring improvements in maintainability and reuse.

3.1 Experimental Setup

The experiments were conducted on a development workstation with the following configuration: Intel Core i7 processor, 16 GB RAM, running Ubuntu 22.04 LTS. We used Visual Studio Code with the Mobioos Forge plugin, version 2024.1, for feature modeling, mapping, and variant generation. The original project management application contained approximately 14,000 lines of Java code organized into 45 main classes across 8 functional modules.

3.2 Feature Mapping Metrics

The first stage of the experiment involved identifying and mapping features to the application's codebase. The results are summarized in Table 1, which reports the number of features, markers, and code elements involved.

Table 1. Feature mapping summary

Metric	Value
Total Features Identified	12
Total Code Files	45
Files with Mapped Features	41
Feature Markers Inserted	61
Code Elements Mapped	230

We achieved a high mapping coverage (>90%), meaning that most of the code elements were successfully linked to one or more features. Features such as Login, Reporting, and Evolution Monitoring had clear and modular implementations, while features like Notification and PEC required more effort due to interdependencies and cross-cutting concerns.

3.3 Variant Configurations and Generation

Using Mobioos Forge, we defined five distinct product configurations based on realistic stakeholder needs. These configurations are listed in Table 2.

Table 2. Sample variant configurations

Variant Name	Enabled Features	Build Size (LOC)	Build Time
Core Only	Login, User Management	7,200	12 sec
Full Edition	All Features	14,100	18 sec
Reporting Edition	Login, Project Tracking, Reporting	9,400	13 sec
Light Edition	Login, PEC, Budget Management	8,300	11 sec
Extended Monitoring	All Core + Evolution Monitoring + Advanced Reporting	12,500	16 sec

The results indicate that the system could reliably produce variants tailored to different use cases. Each variant was compiled and executed to confirm functional integrity. Mobioos Forge's Customization Engine ensured that only the relevant code was included in each variant, thus reducing the size and complexity of deployments.

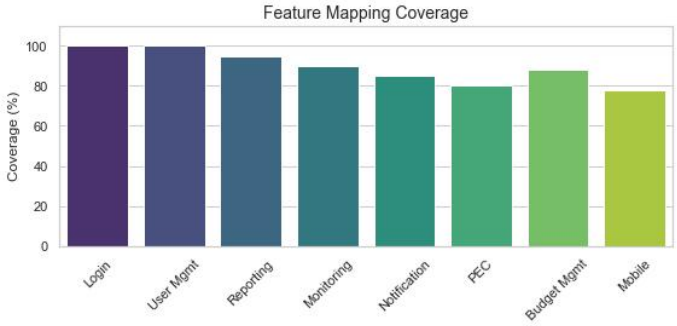

Fig. 2. Feature mapping coverage

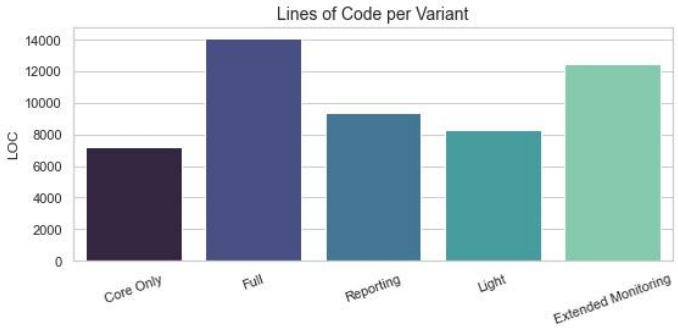

Fig. 3. Lines of Code per variant

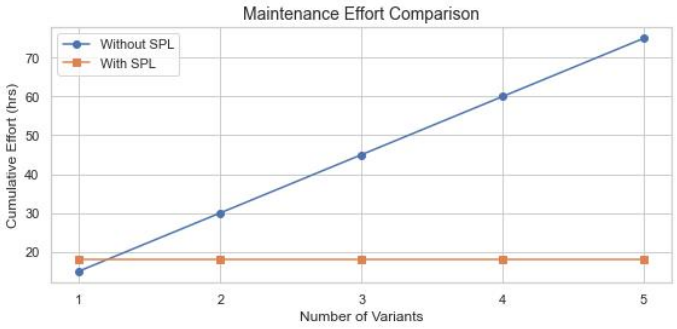

Fig. 4. Maintenance effort comparison

To evaluate software reuse and maintainability improvements due to the SPL transformation, we compared the migrated product line against a baseline scenario where each variant was implemented independently. Key indicators include:

Code Reuse Ratio: Over 70% of the code was reused across all variants, primarily in core modules such as authentication, navigation, and base models.

Maintenance Impact: In a simulated bug-fix scenario (e.g., fixing a login validation issue), a change in the shared component was propagated to all variants automatically. This reduces the defect resolution time and eliminates redundancy.

4 Conclusion

We have presented a case study of migrating a project management application into a software product line using the Mobioos Forge platform. By performing feature modeling, mapping features to code, and using a customization engine, we created a flexible product line capable of generating many variants from one codebase. Quantitative results (feature counts and mapping coverage) and variant examples illustrate the process's effectiveness. We also extended the system with a new microservice-based module, showing how SPL design. Overall, the SPL approach yielded improved reuse, product quality, and reduced maintenance effort. Future work could explore automating feature extraction and further metrics, but this study confirms the practical benefits of SPL migration.

Acknowledgement

This work was supported without any funding.

Conflicts of Interest

The authors declare no conflicts of interest.

References

1. I. M. Murwantara and P. Yugopuspito, "Adaptive Software Management for Economic Drone: Leveraging Software Product Line Engineering for Multi-Mission Efficiency," in 2024 International Conference on Intelligent Cybernetics Technology \& Applications (ICICyTA), 2024, pp. 853–857.
2. D. Ajiga, P. A. Okeleke, S. O. Folorunsho, and C. Ezeigweneme, "The role of software automation in improving industrial operations and efficiency," Int. J. Eng. Res. Updat., vol. 7, no. 1, pp. 22–35, 2024.
3. T. A. O. Fei et al., "makeTwin: A reference architecture for digital twin software platform," Chinese J. Aeronaut., vol. 37, no. 1, pp. 1–18, 2024.
4. A. Guldner et al., "Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green Software Measurement Model (GSMM)," Futur. Gener. Comput. Syst., vol. 155, pp. 402–418, 2024.
5. N. Khoshniat, A. Jamarani, A. Ahmadzadeh, M. Haghi Kashani, and E. Mahdipour, "Nature-inspired metaheuristic methods in software testing," Soft Comput., vol. 28, no. 2, pp. 1503–1544, 2024.
6. M. Coutinho, L. Marques, A. Santos, M. Dahia, C. França, and R. de Souza Santos, "The role of generative ai in software development productivity: A pilot case study," in Proceedings of the 1st ACM International Conference on AI-Powered Software, 2024, pp. 131–138.
7. S. L. Pfleeger and B. Kitchenham, "Evidence-Based Software Engineering Guidelines Revisited," IEEE Trans. Softw. Eng., 2025.
8. O. D. Segun-Falade, O. S. Osundare, W. E. Kedi, P. A. Okeleke, T. I. Ijomah, and O. Y. Abdul-Azeez, "Assessing the transformative impact of cloud computing on software deployment and management," Comput. Sci. \& IT Res. J., vol. 5, no. 8, 2024.
9. L. Kazi and Z. Kazi, "BPriS: Disciplined Agile Delivery Planning Method Based on Work Items List Pattern Applied to Prioritized Semantically Coupled Software Functions Derived from Business Process Model and Software Functional Pattern," Appl. Sci., vol. 15, no. 9, p. 5091, 2025.
10. J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," IEEE Trans. Softw. Eng., 2024.
11. V. D. Kirova, C. S. Ku, J. R. Laracy, and T. J. Marlowe, "Software engineering education must adapt and evolve for an llm environment," in Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, 2024, pp. 666–672.
12. S. Rasnayaka, G. Wang, R. Shariffdeen, and G. N. Iyer, "An empirical study on usage and perceptions of llms in a software engineering project," in Proceedings of the 1st International Workshop on Large Language Models for Code, 2024, pp. 111–118.

13. B. Moroz, A. Saltan, and S. Hyrynsalmi, "Optimizing for Sustainability: Product Ops and Software Waste Reduction," in International Conference on Software Business, 2024, pp. 251–267.
14. D. Ajiga, P. A. Okeleke, S. O. Folorunsho, and C. Ezeigweneme, "Enhancing software development practices with AI insights in high-tech companies," IEEE Softw. Eng. Institute, Tech. Rep. TR-2024-003, 2024.
15. T. M. De Menezes and A. C. Salgado, "Using Logs to Reduce the Impact of Process Variability and Dependence on Practitioners in Requirements Engineering for Traditional Business Process Automation Software," IEEE Access, 2024.

Biographies

1. **Said Naceri** is a Master's student in Computer Science at the Univeristy of Djelfa, Algeria. his research interests include software engineering, SPL, and project management systems.
2. **Walid Kherchofi** is a Master's student in Computer Science at the Univeristy of Djelfa, Algeria. his research focuses on software development methodologies, microservices, and open-source software.

開源應用向軟件產品線的遷移

Said Naceri[1]，Walid Kherchofi[1]

[1]計算機科學系，傑爾夫大學，阿爾及利亞，17000

摘要：軟件復用是現代軟件工程的基石，有助於提升開發效率與系統適應性。軟件產品線（SPL）通過在特定領域利用可復用資產，為創建軟件家族提供了系統性方法。本文介紹了一種將開源項目管理應用借助Mobioos Forge平臺遷移到軟件產品線的方法。同時，我們依據阿爾及利亞近期監管框架（Loi organique18-15），為該應用拓展了一個基於微服務的模塊。結果顯示，其可復用性與靈活性得到提升，這得益於特性建模、變體生成以及模塊化架構。此項工作凸顯了軟件產品線在實際應用中的實用價值。

關鍵詞：項目管理；可變性；軟件產品線；Mobioos Forge

1. Said Naceri，阿爾及利亞傑爾夫大學計算機科學專業的碩士研究生，他的研究興趣包括軟件工程、軟件產品線（SPL）和項目管理系統；
2. Walid Kherchofi，阿爾及利亞傑爾夫大學計算機科學專業的碩士研究生，他的研究重點在於軟件開發方法論、微服務和開源軟件。