Reengineering an Adaptive System to Dynamic Software Product Lines: An Experience report

Dhyego Tavares M. da Cruz
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
dhyegocruz@ufba.br

Nilton F. S. Seixas
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
nilton.seixas@ufba.br

Mayki dos Santos Oliveira
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
maykioliveira@ufba.br

Marcus Elias Silva Freire
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
marcus.elias@ufba.br

Rodrigo R. G. Souza
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
rodrigorgs@ufba.br

Frederico A. Durão
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
fdurao@ufba.br

Cássio V. S. Prazeres
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
prazeres@ufba.br

Ivan do Carmo Machado
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
ivan.machado@ufba.br

Gustavo B. Figueiredo
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
gustavobf@ufba.br

Maycon L. M. Peixoto
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
maycon.leone@ufba.br

Hérsio Massanori Iwamoto
Positivo Tecnologia
Curitiba, Brazil
hersio@positivo.com.br

Eduardo Santana de Almeida
Institute of Computing (IC), Federal University of Bahia (UFBA)
Salvador, Brazil
eduardo.almeida@ufba.br

ABSTRACT

Dynamic Software Product Lines (DSPLs) extend traditional Software Product Lines by enabling runtime adaptation through predefined variability points. Despite their benefits, the migration of legacy or adaptive systems to DSPLs remains underexplored, especially through the use of feature toggles. This study aims to investigate how an adaptive AI-based smart home system can be refactored into a DSPL using feature toggles to manage runtime variability and improve modularity, flexibility, and performance. The research involved (i) a literature review on DSPLs and feature toggle strategies, (ii) the creation of feature models from code analysis, documentation, and expert interviews, and (iii) the incremental implementation of 13 feature toggles across three system modules (RecSys, ChatBot, Vision). Two experiments using factorial design were conducted to assess the impact on performance metrics. The refactored system demonstrated improved adaptability and reduced resource usage without performance degradation. Feature toggles enabled real-time activation/deactivation of functionalities, allowing for user-specific configurations and subscription models. However, limitations were found in toggle dependency management, requiring manual coding solutions. Feature toggles proved to be an effective and practical strategy for transforming an adaptive system into a DSPL, enhancing runtime flexibility and system evolution capabilities. The study highlights key considerations for managing toggles, such as modularization, architectural process, and the importance of documentation.

KEYWORDS

Feature flag, Feature toggle, DSPL

1 Introduction

The Software Product Line (SPL) is a systematic approach that enables the development of a family of software systems based on a shared set of reusable assets [25, 36]. Its primary advantage lies in the reuse of components, which reduces costs, effort, and development time while improving the quality and consistency of products [25]. Furthermore, SPL facilitates customization to address diverse market needs and enhances portfolio scalability. However, adopting an SPL approach can pose challenges, such as the high initial implementation cost, the need for detailed and specialized planning, and the complexity of managing product variability [25]. Additionally, transitioning from traditional methods to an SPL approach may require significant changes in processes and organizational culture, necessitating training and adaptation from the teams involved [26].

Dynamic Software Product Lines (DSPLs) represent a significant evolution from traditional Software Product Lines (SPLs), integrating adaptability into system design, focusing on predefined adaptation points [2]. Unlike conventional SPLs, which generate predefined solutions based on a shared set of functionalities, DSPL-based systems are distinguished by their ability to dynamically adapt at runtime and excel in managing these changes [39]. This enables the software to automatically adjust its functionalities in response to changes in the environment or user requirements, enhancing flexibility and responsiveness. Such capabilities are particularly valuable

in contexts where requirements can shift rapidly, such as mobile systems, IoT applications, and dynamic enterprise environments.

The advantages of DSPLs are considerable. Firstly, their ability to adapt in real-time reduces the need for manual interventions, leading to more efficient and cost-effective system maintenance [1, 29, 30]. Moreover, the advanced customization offered by this approach enhances the user experience by tailoring specific functionalities to individual needs and usage contexts [30, 36, 38]. This not only improves user satisfaction but also provides a competitive advantage in markets characterized by a demand for constant innovation and agility. Furthermore, the scalability and flexibility inherent in DSPL systems enable organizations to address multiple operational scenarios without the need to develop separate products for each case [3, 5, 26].

On the other hand, the absence of a DSPL system can lead to significant limitations. Static systems that cannot adapt to changes in the environment or user preferences may quickly become obsolete, particularly in dynamic industries [15, 26, 29, 38]. Additionally, the lack of flexibility can result in higher operational costs, as updates and modifications require frequent manual interventions [1, 29]. Such systems also struggle to manage variability and meet diverse user demands, compromising both their competitiveness and efficiency [5, 30, 36, 38]. Therefore, adopting a DSPL approach becomes a strategic measure not only to optimize resources but also to ensure the longevity and relevance of the developed systems. To ground this investigation in a practical context, this study focuses on the reengineering of a real-world, adaptive AI-based smart home system. This system is the outcome of a university-industry research and development (R&D) project [6, 10, 11, 19, 24, 35], making it a relevant candidate for exploring the challenges and benefits of migrating a legacy system to a more flexible architecture.

DSPLs are commonly implemented using techniques that enable runtime adaptation, such as variability modeling with feature models and component- and service-based reconfiguration [3, 4, 36]. These techniques employ architectural models to capture the complexity and dynamics of the systems. The models are used to generate code and adaptation policies, which are then executed by middleware. Another strategy that could also be applied is the use of feature toggles, which allow features to be dynamically enabled or disabled without modifying the source code or interrupting system execution [33].

For this purpose feature toggles can offer a practical solution for runtime flexibility, enabling conditional activation or deactivation of features without system rebuilds [8, 14, 21–23, 32, 33]. They streamline development cycles and reduce risks by supporting: **gradual rollout**, introducing new functionality incrementally; **experimentation**, testing feature variations with user groups; and **production control**, disabling features swiftly without deploying new versions [23].

Feature toggles enable real-time dynamism and mutability in code. However, to the best of our knowledge, no prior studies have investigated the use of this technique to fully refactor a system into a dynamic software product line (DSPL). This study proposes the transformation of an existing system into a DSPL by adopting feature toggles as the primary mechanism for implementing variability points. **While the primary goals are to enhance modularity and runtime flexibility, it is critical to validate that these benefits do not come at a prohibitive performance cost**. Therefore, to assess the practical implications and non-functional impact of this approach, a performance evaluation was conducted to compare system behavior before and after the integration of feature toggles.

The remainder of this paper is organized as follows: Section 2 introduces the concepts of feature toggles. Section 3 reviews related work on smart homes and the intersection of DSPL with feature toggles. Section 4 details the research design, including the creation of feature models and the implementation of toggle management. Section 5 explains the experimental design for performance evaluation. Section 6 presents the preliminary results from the experiments. Section 7 discusses the findings, benefits, and challenges encountered. Finally, Section 8 concludes the paper and outlines future work.

2 Feature Toggles

Feature Toggles are a technique that allows developers to enable or disable a feature or specific part of the code [22], offering control over code behavior without requiring a system redeployment in a *runtime execution*, as shown in Listing 1. This approach facilitates the incremental integration of new features and experiments, even when the features are not fully ready for release or when developers need to deliver them to a specific group of users.

The code in Listing 1 demonstrates a basic implementation of a feature toggle using the Unleash[1] tool feature toggle management system to control the activation of a specific functionality at runtime. In this example, the *function()* checks the status of a feature toggle *toggle_name* using the *feature.enable* method, which accepts the toggle name and a *fallback* option for default behavior. If the toggle is not enabled, the code inside the *if* block runs, representing the *current implementation* or the new functionality. If the toggle is disabled, the code in the *else* block executes, reflecting an alternative or previous implementation. Both blocks return the respective implementation based on the toggle's status.

```
def function():
    if not feature.enable('toggle_name', fallback):
        #current implementation
        return implementation
    else:
        #another current implementation
        return another current implementation
```

Listing 1: Example of feature toggle in a method

Furthermore, feature toggles can be applied in continuous development and A/B testing environments, facilitating real-time adjustments and risk mitigation by quickly disabling problematic features [23]. This work contributes to the growing body of research that highlights the potential of feature toggles as a flexible, scalable tool to support the high adaptability required in DSPLs, particularly in smart home applications.

[1] https://github.com/Unleash/unleash/tree/main

3 Related Work

To explore this intersection, we divided this section into two parts: the first subsection discusses studies in the smart home domain—the context of our system—while the second subsection highlights research connecting DSPL with the feature toggle methodology.

3.1 Smart Home Domain

This present study aims to adapt a recommendation system for smart homes, equipped with artificial intelligence, into a dynamic software product line, based on approaches described in the literature [6, 19]. This system seamlessly integrates with real smart devices, learns user behavior patterns, and strives to deliver enhanced, personalized recommendations for these devices.

Some works discuss innovative solutions and challenges in the field of smart homes and DSPL [7–9, 35]. The application of feature models and executable reconfiguration plans stands out as a strategy to adapt systems to changes in the environment and user actions, ensuring runtime compatibility [8]. Additionally, the use of Aspect-Oriented Programming (AOP) in DSPLs has demonstrated advantages in modularity and coupling, although it requires attention to the propagation of changes [7]. In terms of device integration and artificial intelligence, distributed architectures with cloud-based services have proven effective in learning user habits to recommend actions that optimize comfort and energy consumption [19]. Other solutions, such as the Rudas system, emphasize energy efficiency and remote control in IoT networks [9].

Cetina et al. [8] paper proposes a model-based approach for the dynamic reconfiguration of smart homes, utilizing feature models to specify how the system can evolve. The approach enables smart homes to self-configure in response to changes in user activities and the physical environment. Feature models are used to define the boundaries within which the system can evolve, avoiding technical details and specifying reconfiguration possibilities declaratively. The system includes a Context Monitor that assesses contextual conditions, a reconfigurator that leverages resolutions associated with these conditions to query the feature model, and a Reconfiguration Plan to modify the system architecture. The approach also incorporates model-based validation to analyze configurations and ensure specific properties.

Complementing these findings in Carvalho et al. [7] work, the feature model was employed as a central tool to define and manage reconfiguration scenarios for a smart home environment, enabling the specification of conditions for dynamic system variations. This approach aligns with the principles of DSPL engineering by using feature models to support runtime adaptations. The reconfiguration points identified in the study were strategically categorized into three key domains: security, lighting, and temperature control. These categories illustrate the focus on adaptability and evolution in system functionality, addressing the need for tailored variations to meet specific runtime conditions and requirements.

3.2 Feature Toggles and DSPL

Numerous studies have investigated the application of feature toggles in software development [12, 21, 22, 28, 32, 33]. Among these, Meinicke et al. [28] conducts a comparative analysis of two distinct approaches, elucidating their technical similarities, differences, and potential avenues for cross-disciplinary learning based on insights gathered from expert interviews. Similarly, Mahdavi-Hezaveh et al. [22] examines the advantages of feature toggles in the context of continuous integration and delivery, while also emphasizing the risks associated with their misuse. Mahdavi-Hezaveh et al. [22] identified 17 industry best practices for the effective management of feature toggles, derived from a qualitative analysis of both gray literature and peer-reviewed publications. A notable finding is the prevalent adoption of specialized management systems, such as LaunchDarkly [20] and Split [34], which have become integral tools in this domain.

Expanding on this, Mahdavi-Hezaveh et al. [21] proposes heuristics and metrics to improve feature toggle practices, aiming to reduce complexity, enhance maintainability, and limit technical debt. The study emphasizes that toggles should be used judiciously, be self-descriptive, avoid code duplication, and be removed when obsolete. Two core practices are recommended:

- **Management Practices:** Implementing a feature toggle management system is crucial for organizing and reducing code complexity.
- **Clean-up Practices:** Regularly removing unused feature toggles is essential to prevent dead code, control complexity, and minimize technical debt.

In another study Jézéquel et al. [18] explores the integration of Feature Toggles with Software Product Lines (SPL). The study highlights how Feature Toggles provide a runtime variability resolution alternative for managing multiple feature branches in the source code. The authors propose a unified approach that models variability using a feature model, enabling partial resolution during the design phase and activating toggles at runtime. They demonstrate this concept through a toy example and a configurable authentication system, interacting with popular frameworks such as Togglz[2]. The article discusses the advantages and disadvantages of various implementation strategies, emphasizing the need for structured patterns to effectively manage variability in major programming languages like Java and C.

In this work, we explored the integration of feature toggles within smart home environments to DSPLs. Unlike previous studies that focused on using predefined feature models for system reconfiguration based on user or environmental changes, our approach dynamically manages feature activation and deactivation at runtime, without requiring significant architectural modifications. As noted by Weyns et al. [39], runtime adaptation can be challenging, sometimes leading to instability and inefficiency; however, feature toggles offer a solution by enabling flexibility in system behavior, allowing for rapid modifications and incremental feature implementations [23].

4 Research Design

In this section, we delve into the methodology employed to execute the refactoring process of our system. This approach is structured into two key parts. First, we provide a detailed description of how we designed the system's feature architecture using feature models, highlighting the principles and strategies that guided this design. In

[2]https://www.togglz.org/

Table 1: Description of the Features and Interactions of RecSysModule

Feature	Function	Interactions
Intelligence	Responsible for all the intelligence of the module	UsageHistory
EnvironmentBehavior	Controls the behavior of the devices in the environment	UsageHistory
EventPrevision	Forecasts possible events in the environment	UsageHistory
Anomaly	Detects an anomaly in the usage of devices	UserNotification, HumanizedMessage, UsageHistory
ActionRecommendation	Recommends actions for the devices	HeuristicRules, HumanizedMessage
UsageHistory	Records the usage of the devices	Intelligence
HumanizedMessage	Transcribes low-level messages to high-level messages	ActionRecommendation, Anomaly
HeuristicRules	Set of rules for handling recommendations	EnvironmentBehavior
UserNotification	Generates notifications for users	HeuristicRules
RecommendationSuggestion	Notifies users with recommendation suggestions	Intelligence, UsageHistory, HumanizedMessage
UncommonPresence	Notifies of uncommon presence in the environment	Intelligence, UsageHistory, HumanizedMessage
DeviceAnomaly	Notifies detected anomalies in devices	Intelligence, UsageHistory, HumanizedMessage

the second part, we explore the feature management techniques implemented to control feature toggles, discussing the tools, practices, and mechanisms that ensure effective feature handling throughout the refactoring process.

In this study, we refactored an artificial intelligence (AI) system originally structured into three main modules, aiming to improve its overall efficiency and organization: the **RecSysModule**, which includes essential features such as device control, usage tracking, heuristic rules, devices anomaly detection, and event prediction; the **ChatBotModule**, designed for direct user interaction with the system and message logging; and the **VisionModule**, which provides the system with vision-based capabilities, including facial recognition, object detection, intruder detection, and security management. In addition to the three main modules, our system includes a broker that manages communication between modules and handles inputs from the smart home.

The transition to a dynamic system was carried out through a carefully structured three-step process: (1) an in-depth literature review to provide a solid foundation and guide the overall approach, (2) the creation of a detailed model to organize and represent how features were structured within the system using feature models, as illustrated in Figure 1, and (3) the implementation of feature toggles derived from the developed feature models to enable dynamic control and flexibility. Each of these steps is elaborated in greater detail in the subsequent sections.

4.1 Literature Review

This initial phase of article search was of fundamental importance, as it allowed us to identify and evaluate development techniques that would facilitate the system transition. Our primary objective was to transform the existing system rather than rebuild it from scratch. Following this stage, we were able to determine the most suitable technique to apply in our work and conduct a comparative analysis with other methodologies to validate its appropriateness and effectiveness.

The first step in acquiring related studies was conducting a review of the state of the art, focusing on DSPL. This process involved searching prominent conferences such as VaMoS, SPLC, SEAMS, GPCE, ICSE, FSE, ASE, and SBCARS, as well as renowned journals like JSS, IST, IEEE TSE, and ACM TOSEM. The searches were performed using the DBLP indexer, covering relevant publications from 2005 onward. This initial effort aimed to map recent advancements and identify significant contributions to the field.

To refine the scope, the study focused on research where "feature toggle" was the central theme. Searches were conducted in widely recognized databases, including ACM Digital Library, IEEE Xplore, and ScienceDirect, using terms such as "feature toggle," "feature flag," and similar keywords. These terms were derived from trusted sources, including Martin Fowler's blog and the work of Hezaveh et al. [14, 22]. Following this stage, duplicate results were removed, and a "snowballing" technique [40] was employed to identify additional relevant studies by analyzing references from the collected articles.

This literature review process not only mapped the intersection of feature toggles and DSPL but also provided a deeper understanding of academic progress in the field. It offered insights into best practices and methodological choices discussed in the literature and most used tools, providing a broader and more grounded view of the academic contributions to the use and development of feature toggles.

4.2 Design of Feature Models

The transition from the existing adaptive system to a DSPL managed through feature toggles entailed an incremental and carefully planned refactoring process. The primary objective was not only to incorporate runtime variability capabilities but also to ensure that this adaptation occurred in a manner that minimized system complexity and overhead. This process involved the identification of variability points, their mapping to feature toggles as illustrated in Figure 1, and the gradual integration of these mechanisms across the system's three core modules: RecSys, ChatBot, and Vision. The implementation of 13 feature toggles required a thorough analysis of the legacy codebase and the application of targeted refactoring strategies to accommodate the newly introduced variability

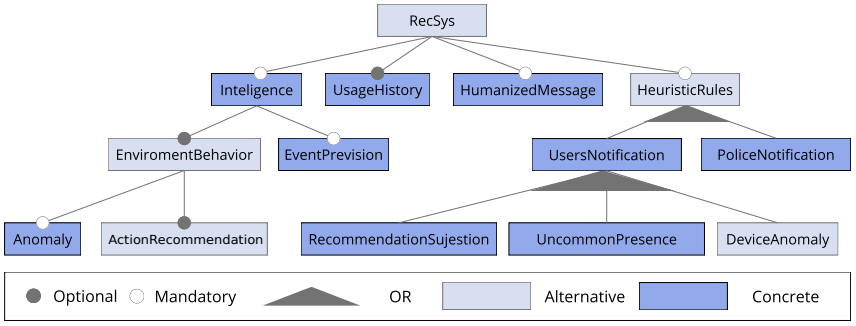

Figure 1: Example of feature model for the RecSys Module.

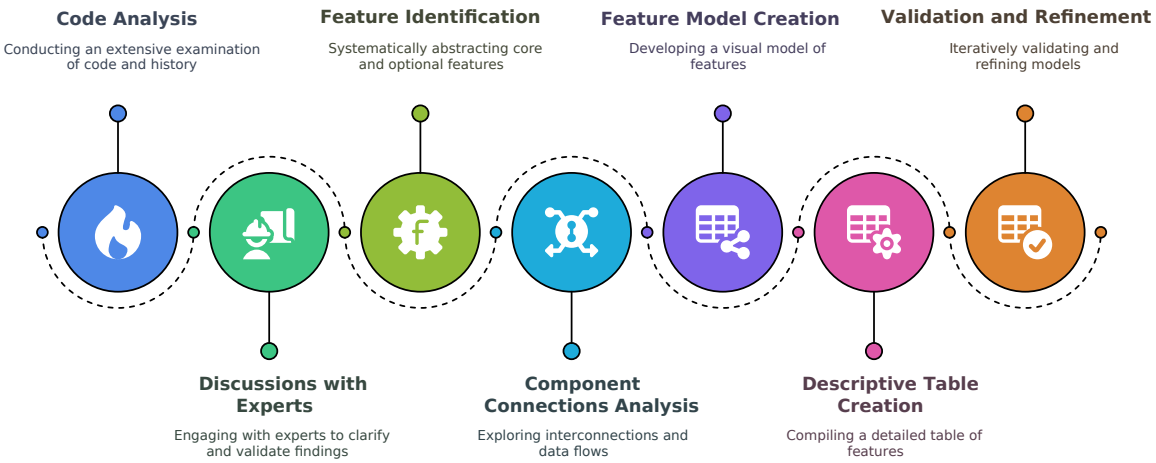

Figure 2: Roadmap of processes followed before refactoring

mechanisms. Table 1 complements this by offering a comprehensive description of the features present within the **RecSysModule**. Together, these resources serve as valuable tools for understanding the design and functionality of the recommendation system.

Through these discussions and code reviews, we gained valuable clarity on the interconnections between components, uncovering features that were distributed across multiple sets of classes rather than confined to specific methods. This exploration enabled us to construct a detailed feature model, representing the implemented features while highlighting potential points of variability. The resulting model provided a clear and structured visualization, serving as a crucial guide for the dynamization process and ensuring a cohesive approach to enhancing the system's adaptability and flexibility.

In summary, we follow the steps below to design and structure the system before refactoring. These steps can also be visualized in a simplified way in the Figure 2:

- **Code Analysis:** An extensive examination of the project repositories, focusing on both the source code and the version control history. This analysis aimed to map implemented functionalities, identify code patterns, and uncover hidden dependencies not explicitly documented. Static code analysis tools were employed to facilitate the detection of architectural components, class hierarchies, and potential technical debt areas that could impact the dynamization process.
- **Discussions with Experts:** Structured interviews and focused workshops with developers, system architects, and domain practitioners. These interactions were designed to clarify doubts arising from documentation and code reviews, gain insights into design rationales, and understand the practical challenges faced during development and maintenance. This step also helped validate our initial findings and refine our understanding of complex system behaviors.
- **Feature Identification:** Systematic abstraction of functionalities based on insights gathered from documentation, code analysis, and expert discussions. The focus was on identifying core features, optional capabilities, and potential variability points, particularly within complex modules like the recommendation system. This process involved defining clear feature boundaries and dependencies, ensuring that all relevant aspects of the system were captured.
- **Component Connections Analysis:** In-depth exploration of the interconnections between system components, emphasizing how functionalities are distributed across different classes and modules. This step involved tracing data flows,

control flows, and interaction patterns to understand how disparate components collaborate to achieve system objectives. Special attention was paid to cross-cutting concerns that might influence the dynamization strategy.

- **Feature Model Creation:** Development of a detailed visual model (Figure 1) to represent the identified features and their interrelations. This feature model provided a structured framework for visualizing the system's functional architecture, highlighting feature hierarchies, dependencies, and variability points. The model served as a reference for guiding design decisions during the refactoring process.
- **Descriptive Table Creation:** Design and compilation of a comprehensive table (Table 1) that details the features of modules. Each feature was described in terms of its functions, implementation details, and interactions with other system components. This table complemented the feature model by offering a granular, text-based representation of the module's capabilities.
- **Validation and Refinement:** An iterative validation process to ensure the accuracy and completeness of the identified features and their representations. This involved revisiting documentation, reanalyzing code, and conducting follow-up discussions with experts to address ambiguities or gaps. Feedback from these activities was used to refine both the feature model and descriptive table, ensuring their reliability as tools for supporting the dynamization process.

4.3 Feature Toggle Management

Given the limited timeframe for implementation, it became imperative to define a strategy to manage system variability effectively. The adoption of a feature toggle management tool emerged as an efficient approach to optimize the achievement of our objectives and is strongly recommended in the literature [23]. The use of a dedicated tool would simplify the development process by eliminating the need to design a custom control interface and reducing the architectural planning overhead. For an existing legacy system such as ours, developing a new tool from scratch would entail prohibitive costs, making this approach a more cost-effective solution for achieving our goals.

To select the most appropriate tool, we established three key requirements: it had to be **open-source**, easily installable and **portable**, and, most importantly, provide support for **user management**. Open-source tools offer transparency and a cost-effective alternative. Portability allows for rapid deployment across different environments and ensures platform compatibility. Support for user management was the most critical criterion, as the goal of the system was to enable different configurations per user type, allowing for scenarios such as subscription-based models. To identify suitable candidates, we reviewed existing literature and conducted additional searches for feature toggle repositories on GitHub.

Among the tools considered—Unleash, LaunchDarkly, Split, PostHog, GrowthBook, and Flagsmith [13, 16, 20, 31, 34, 37]—Unleash emerged as the most suitable option. This decision was supported by the findings of Mahdavi-Hezaveh et al. [23], which demonstrated the effectiveness of Unleash as well as its alignment with the three predefined criteria. The tool provided the necessary features, such as flexible activation strategies and access control, within an open-source framework, thereby aligning with the technical and practical objectives of this study.

4.4 Challenges and Mitigation Strategies

Refactoring a legacy system to incorporate feature toggles was not without challenges. The primary obstacle lay in identifying and managing the intrinsic dependencies among existing functionalities. The source code, developed over an extended period, exhibited a high degree of coupling, which hindered the clean extraction of individual features to be controlled via toggles.

Challenge 1: Code Coupling and Identification of Variability Points. The analysis of the legacy code revealed tightly interwoven functionalities, making it difficult to define clear boundaries for toggles. To address this, we adopted a top-down impact analysis approach, starting with high-level functionalities and drilling down into affected subcomponents. Intensive pair programming sessions and code reviews were conducted to map method and class invocations, thereby identifying the most appropriate injection points for toggles.

Challenge 2: Management of Toggle Dependencies Although tools such as *Unleash* simplify individual toggle activation and deactivation, we encountered limitations in explicitly managing logical dependencies among toggles (e.g., *Feature A* can only be activated if *Feature B* is active). To circumvent this limitation, we implemented manual coding solutions. Specifically, we developed an internal orchestration module that, by querying the state of toggles in *Unleash*, applied predefined validation rules to ensure configuration consistency. For instance, a validation service was created to verify whether the *User Preferences Database* toggle `toggle_user_preferences_db` was active before allowing the activation of the *Advanced Personalization* toggle `toggle_advanced_personalization`. This module functioned as a safety and consistency layer, compensating for the lack of native support for managing complex dependencies within the tool.

Challenge 3: Validation and Testing. Introducing runtime variability increased the complexity of testing scenarios. To address this, we developed a comprehensive test matrix that considered all logically possible toggle combinations for each module. Automated integration tests were adopted to simulate different user profiles and their respective active toggle configurations.

4.5 Lessons Learned and Applicable Knowledge

The experience of refactoring an existing adaptive system into a DSPL using feature toggles provided valuable insights for future legacy system migrations:

Feature Toggle Mapping. The modeling of features and their mapping to toggles must be exhaustive and involve domain experts. Underestimating the complexity of existing dependencies within the legacy codebase can lead to significant integration challenges.

Effort and Planning. Incremental refactoring is essential. However, the effort required for adapting the source code and creating variability points must be carefully planned and allocated.

Dependency Management. The absence of native support for complex toggle dependencies in some feature management tools may necessitate the implementation of a custom orchestration

layer. While this introduces additional complexity, it is crucial for maintaining functional consistency within the system.

Importance of Testing. The testing strategy must be robust enough to cover the numerous toggle combinations. Automation and end-to-end integration tests are indispensable to ensure system stability across all variability scenarios.

Benefits Beyond Performance. Although we validated the absence of significant performance degradation, the most notable benefits were observed in the agility of releasing new functionalities, the personalization of user experience, and the improvement of system maintainability through modularization.

5 Experiment Design

The main goal of refactoring the adaptive smart home system into a Dynamic Software Product Line (DSPL) was to improve modularity and flexibility, enabling runtime variability to support diverse user configurations and subscription models. While the focus was on enhancing configurability and maintainability, it was also essential to assess whether the introduction of feature toggles would lead to unacceptable performance degradation. To ensure the increased flexibility did not come at a prohibitive computational cost, we conducted a rigorous performance evaluation to validate the practical feasibility of our approach.

To evaluate the performance implications, we conducted two independent full factorial experiments [17]. This approach allows for the analysis of both main and interaction effects between controlled factors. The response variables under study include **execution time per request**, **CPU usage**, and **memory consumption**.

The experiments were designed to analyze the following factors:

- **Toggle Activation (Factor A in Exp. 1):** This factor compares the system's performance with and without the feature toggle mechanism. The levels are `True` (the refactored system with toggles active) and `False` (the original legacy system).
- **User Plan (Factor A in Exp. 2):** Made possible by the toggle infrastructure, this factor investigates performance based on different feature sets available to a user. The levels are `Full` (all features available) and `Basic` (selected computationally intensive features, like facial recognition, are disabled).
- **Scenario (Factor B in both Exp.):** This factor introduces environmental variability by simulating two distinct household profiles, `House 1` and `House 2`, which represent different user behaviors and device interaction patterns.

For each experiment, the general linear model used to analyze the response variable Y is:

$$Y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B \quad (1)$$

Where:

- Y is the response variable (e.g., execution time, memory usage).
- x_A and x_B are the levels of the factors being analyzed in a given experiment (e.g., User Plan and Scenario).
- q_0 is the grand mean (overall average).
- q_A and q_B are the main effects of the factors.
- q_{AB} is the interaction effect between the factors.

This factorial design provides a rigorous framework for quantifying the individual and combined effects of toggle activation and user configuration on system performance. All tests were conducted using synthetic user behavior data generated by the HESTIA framework, ensuring control and reproducibility.

Table 2: Experiment Matrix: Toggle Activation, Scenario, and User Plan

Experiment	Run	Toggle	Scenario	User Plan
1	1	True	House 1	Full
1	2	True	House 2	Full
1	3	False	House 1	(Legacy)
1	4	False	House 2	(Legacy)
2	5	True	House 1	Basic
2	6	True	House 2	Basic
2	7	True	House 1	Full
2	8	True	House 2	Full

5.1 Experimental Setup

Each of the eight experimental conditions listed in Table 2 was applied to all three system modules (RecSys, Vision, and ChatBot), resulting in a total of 24 runs. In each run, 300 requests were processed per module, simulating user behavior in the house. Runs 1–4 correspond to Experiment 1 (toggle vs. legacy), while Runs 5–8 represent Experiment 2 (Full vs. Basic plan, with toggles enabled).

To introduce environmental variability, two household scenarios were simulated, each representing different user behavior and interaction patterns with smart devices:

- **House 1**[3]**:** A single resident primarily using the bedroom, interacting with lights, motion sensors, and smart plugs. The routine is irregular, involving computer use and common household tasks, with more interaction during weekends.
- **House 2**[4]**:** Two residents with structured and diverse routines. One focuses on work-related computing activities with scheduled breaks; the other alternates between music, device usage, and household tasks. Activity is more dispersed across rooms and time slots.

6 Preliminary Evaluations

The project focused on increasing system flexibility to easily toggle features, enabling tailored experiences for different customer segments from a single codebase. This approach ensures seamless updates without requiring new downloads, delivering value to customers and the company. Additionally, this approach could provide the company with a subscription model that would ensure recurring revenue, contributing to a stable and predictable financial flow.

In Figure 3, we can see a comparison of the system's modules—Chat, RecSys, and Vision—with and without the use of feature toggles, considering execution time, CPU usage, and memory usage. The results indicate that the use of toggles slightly affects performance negatively, and in some cases, it has virtually no effect. For the Chat and RecSys modules, the execution time remains relatively stable regardless of toggle usage. However, in the Vision module, a slight increase is observed when toggles are enabled, slightly

[3]Dataset available at: https://github.com/hestia-sim/datasets/tree/main/1_resident_1_room
[4]Dataset available at: https://github.com/hestia-sim/datasets/tree/main/2_resident_6_room

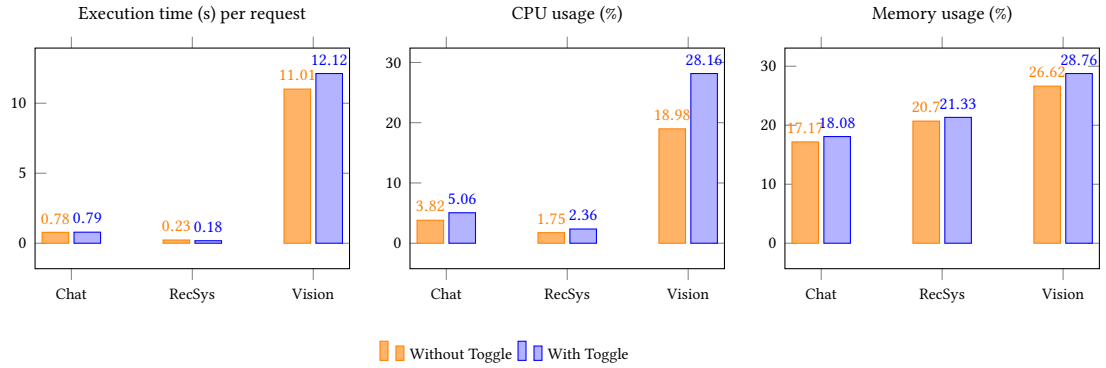

Figure 3: Comparison of modules with and without toggle across execution time, CPU and memory usage.

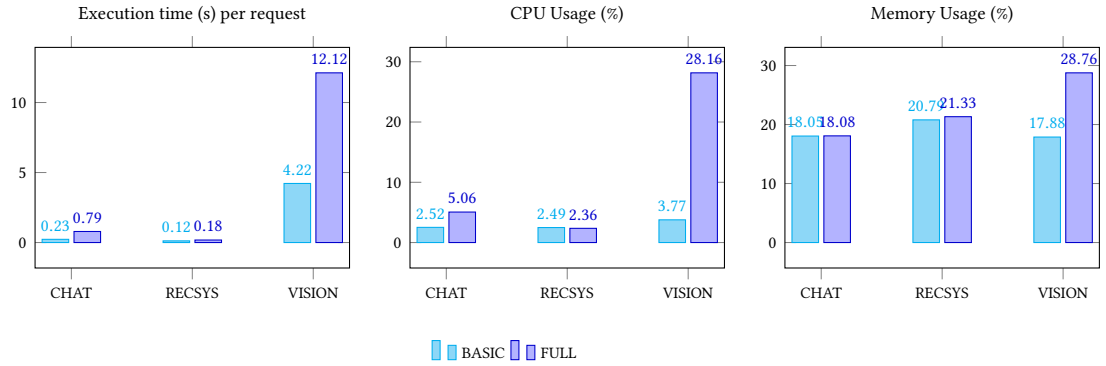

Figure 4: Comparison of BASIC and FULL profiles across modules in terms of execution time, CPU and memory usage.

worsening the time from 11.01 seconds to 12.12 seconds, which indicates a 9.15% degradation in response time per request. Regarding CPU usage, the Chat and RecSys modules show a lower percentage when toggles are not applied. In the Vision module, CPU usage is also increased with toggles (18.98% without toggle compared to 28.16% with toggles). Regarding memory usage, all modules present a small increase when toggles are enabled, with the most significant difference in the Vision module (28.76% with toggles versus 26.62% without toggles), which overall can be considered an insignificant increase in memory consumption.

Figure 4 compares the BASIC and FULL usage profiles across the same modules and metrics. As expected, the FULL profile demands more resources than the BASIC one in all modules. For the Chat module, execution time (0.23 seconds up to 0.79 seconds in FULL profile) and CPU usage show little differences between the profiles (2.52% in BASIC profile to 5.06% in FULL profile). For the RecSys module, we did not identify such representative differences when applying different user profiles, only a small reduction in memory consumption (21.33% down to 20.79 in BASIC profile). In contrast, the Vision module experiences a noticeable decrease in all metrics. In execution time, when implemented in the BASIC profile, a high decrease was observed (12.12 seconds in FULL profile down to 4.22 seconds in BASIC profile). In addition, CPU usage in the Vision module demonstrated a remarkable reduction under the BASIC profiles (28.16% in FULL profile down to 3.77% in BASIC profile). Memory usage also reduced consistently in the BASIC profile across the Vision module (28.76% in FULL versus 17.88% in BASIC profile).

Overall, the Vision module is the most resource-intensive across all metrics. The results suggest that feature toggles have minimal or even negative effects on system performance. In contrast, applying the BASIC profile led to a sharp drop across all metrics, confirming the effectiveness of using feature toggles as a runtime configuration mechanism that enables user-specific adaptations and resource optimizations without altering the architecture. While certain performance improvements were observed—particularly under the BASIC profile—these are to be interpreted as byproducts of functionality scoping rather than as primary objectives of the refactoring

Furthermore, when comparing the BASIC profile with the configuration without feature toggles, it is evident that the BASIC plan offers a beneficial trade-off by reducing resource consumption since prior to the refactoring and implementation of feature toggles the legacy system was executed entirely in a *"FULL"* profile state, maintaining acceptable performance levels.

Figure 5 presents the computed effects of each factor (A: User Profile, B: Scenario) and their interaction (AB) on the response variable execution time per request, using the sign table method described by Jain [17]. It is evident that the Vision module exhibits

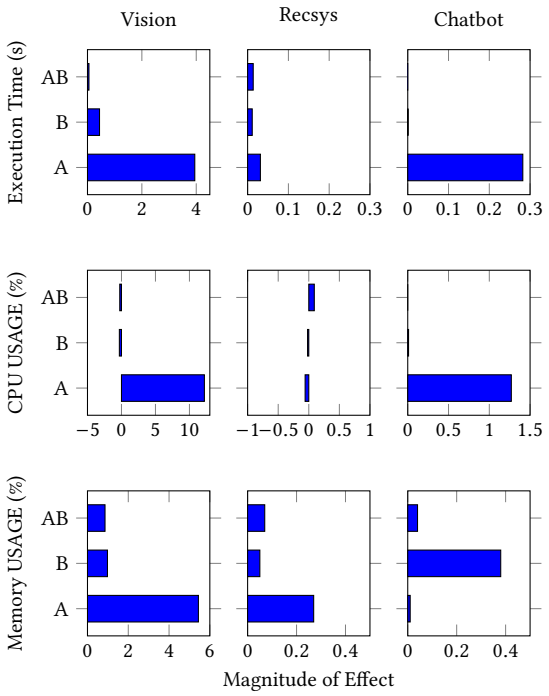

Figure 5: Interaction graph showing the relationship between factors and response variables: Execution Time Per Request, CPU Usage, and Memory Usage. The three sets of graphs represent the effects of the interactions of the factors.

the highest sensitivity to the user plan configuration, with a main effect of approximately 3.95 seconds. This indicates a substantial increase in execution time when switching from the BASIC to FULL profile, primarily due to the activation of computationally intensive features such as facial recognition and weapon detection.

Although the Recsys and Chatbot modules demonstrate lower absolute execution times, the user profile still emerges as the dominant factor in both, reaffirming its cross-cutting influence on resource consumption. The interaction effect (AB) is negligible across all modules, suggesting that the influence of the scenario (House 1 or House 2) does not significantly alter the impact of the user profile. These findings support the rationale for incorporating feature toggles into legacy systems: enabling tailored configurations per user profile not only provides more flexible service offerings but also enables resource optimization and cost efficiency. This adaptability is particularly valuable for scaling services across diverse usage patterns while maintaining performance targets.

Regarding CPU usage, a similar trend is observed: the Vision module is again the most affected by the user profile (factor A), with a substantial positive effect of approximately 12.2%, indicating that enabling the FULL profile significantly increases CPU consumption. Conversely, for the Recsys and Chatbot modules, the effects are minimal and oscillate around zero, reinforcing the idea that these modules are not CPU-bound under the tested configurations. The scenarios (factor B) and its interaction with the user profile (AB) show negligible effects across all modules, suggesting that CPU demands are primarily driven by the activation of features tied to the user plan rather than environmental or contextual variables.

In terms of memory usage, the Vision module again exhibits the largest sensitivity, with a 5.44% increase linked to the FULL profile. This aligns with its reliance on memory-intensive tasks such as image processing and deep learning model inference. For Recsys and Chatbot, the impact of the user profile is comparatively smaller, but still present, particularly in Recsys where both factors A and B contribute modestly. Notably, in the Chatbot module, factor B (scenario) leads to a higher memory usage than factor A, which may be attributed to scenario-specific dialogue states or context caching mechanisms.

Overall, the results confirm that user profile configuration (A) is the primary driver of resource usage across all modules, with the Vision module being the most resource-intensive. The minimal impact of scenario (B) and interaction (AB) effects underscores the predictability of resource consumption patterns, supporting the assertion that feature toggle favored the reduction of the overall system load from the moment the BASIC profile was added to the system.

7 Discussion

Following the mapping of system functionalities and the construction of a feature model, 27 features were identified across three core modules. Thirteen of these features were selected and implemented as feature toggles, introducing runtime variability points and enabling the transformation of the system into a more dynamic and configurable architecture aligned with DSPL principles.

The results presented in the Preliminary Evaluations section provide empirical evidence of the practical impact of this approach. The toggles enabled the configuration of distinct user profiles—namely, FULL and BASIC—resulting in tangible differences in system behavior and resource consumption. Notably, the Vision module exhibited a substantial improvement in performance when operated under the BASIC profile, with execution time per request reduced by approximately 65% (from 12.12s to 4.22s), CPU usage lowered by 24.39 percentage points (from 28.16% to 3.77%), and memory usage reduced by over 10%. These findings support the assertion that feature toggles not only enhance adaptability but also facilitate effective resource optimization based on usage context [14, 22].

Across all modules, user profile (i.e., toggle configuration) emerged as the most influential factor in determining performance, outweighing the variability introduced by distinct usage scenarios. This reinforces the effectiveness of runtime variability control through toggles, especially in adaptive systems where tailored feature sets are required for different user types or operational contexts.

Moreover, the deployment of toggles contributed to architectural modularization and maintainability. By decoupling critical functionalities into independently manageable toggle points, the codebase became more flexible to updates, testing, and rollback operations. This modularization aligns with best practices in DSPL engineering [2, 4], where runtime adaptation must be carefully orchestrated to prevent system instability or unintentional side effects.

Despite these benefits, the study also identified challenges. The toggle management tool employed lacked native support for feature

dependencies, requiring manual coordination to preserve consistency—particularly in complex modules such as RecSys. This limitation highlights the need for more advanced toggle management mechanisms in DSPL contexts, especially when feature interdependencies are nontrivial.

Additionally, the absence of clearly defined user personas and the partial documentation of legacy components limited the potential for full system dynamization. Without explicit stakeholder requirements for user segmentation, customization efforts remained constrained to internal code-level adjustments, rather than being driven by external configurability.

From a DSPL lifecycle perspective, the implementation of toggles introduced concerns related to technical debt—a well-known issue in toggle-based development [22, 27]. Although the toggles in this study were designed as permanent variability mechanisms, standard practices for managing them were adopted to prevent long-term code degradation. These included consistent naming conventions, toggle usage monitoring, comprehensive testing of feature interactions, and the use of dedicated management tools to ensure traceability and maintainability [14, 22].

In summary, this experience report confirms the feasibility and utility of adopting feature toggles as a central mechanism in refactoring legacy adaptive systems into DSPLs. While the toggle-based strategy introduced runtime flexibility, resource control, and architectural decoupling, it also brought to light critical considerations around tooling, documentation, and the need for structured governance. These findings offer valuable insights for practitioners and researchers aiming to evolve existing systems into DSPLs, particularly in domains requiring real-time adaptability, such as smart environments and context-aware applications.

8 Conclusion

This article aimed to refactor an adaptive system into a DSPL. To address this, feature toggles emerged as a key strategy, enabling real-time activation and deactivation of functionalities and transforming a static system into a dynamic one. By analyzing system interconnections and creating a feature model, 13 variability points were identified, enhancing modularity and agility in addressing user needs. The study contributes to DSPL research by demonstrating how feature toggles can adapt systems not originally designed for dynamic environments. It highlights the importance of modularization and configuration management for successful adaptive systems and provides practical insights for organizations to improve system flexibility and user experience. Future work includes a quantitative evaluation of feature toggles' impact and developer interviews to further assess their benefits and challenges.

As part of our future work, we plan to build on the results presented in this article by testing the system in a real-world environment with actual users. This will allow us to validate its behavior outside of a simulated context and assess its performance under practical conditions.

Additionally, we aim to conduct a comparative analysis between the feature toggles methodology and alternative approaches. This comparison will help us gather more concrete, data-driven insights into how the system behaves across different scenarios and under varying conditions. It is worth emphasizing that, although variations in performance were observed, the core contribution of this refactoring effort lies in establishing a flexible and modular architecture that supports dynamic system evolution, rather than performance gains.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. It was also supported in part by the FAPESB INCITE PIE0002/2022 grant, and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, grant #403231/2023-0.

REFERENCES

[1] Emad Albassam, Hassan Gomaa, and Daniel A Menascé. 2017. Variable recovery and adaptation connectors for dynamic software product lines. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*. 123–128.

[2] Mahdi Bashari, Ebrahim Bagheri, and Weichang Du. 2017. Dynamic software product line engineering: a reference framework. *International Journal of Software Engineering and Knowledge Engineering* 27, 02 (2017), 191–234.

[3] Davide Basile, Maurice H Ter Beek, Felicita Di Giandomenico, and Stefania Gnesi. 2017. Orchestration of dynamic service product lines with featured modal contract automata. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*. 117–122.

[4] Nelly Bencomo, Peter Sawyer, Gordon S Blair, and Paul Grace. 2008. Dynamically adaptive systems are product lines too: using model-driven techniques to capture dynamic variability of adaptive systems.. In *SPLC (2)*. 23–32.

[5] Gunnar Brataas, Svein Olav Hallsteinsen, Romain Rouvoy, and Frank Eliassen. 2007. Scalability of decision models for dynamic product lines. In *11th International Conference, SPLC 2007, Kyoto, Japan, September 10-14, 2007*.

[6] Denivan Campos, Luana Martins, Joselito Mota, Dhyego Tavares, Jander Pereira, Mayki Oliveira, Denis Boaventura, Diego Correa, Eduardo Ferreira, George Pinto, et al. 2024. Designing, Implementing, and Testing AI-Oriented Smart Home Applications: Challenges and Best Practices. In *European Conference on Software Architecture*. Springer, 83–99.

[7] Michelle Larissa Luciano Carvalho, Matheus Lessa Goncalves Da Silva, Gecynalda Soares da Silva Gomes, Alcemir Rodrigues Santos, Ivan do Carmo Machado, Magno Luã de Jesus Souza, and Eduardo Santana de Almeida. 2018. On the implementation of dynamic software product lines: An exploratory study. *Journal of Systems and Software* 136 (2018), 74–100.

[8] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. 2009. Using feature models for developing self-configuring smart homes. In *2009 Fifth International Conference on Autonomic and Autonomous Systems*. IEEE, 179–188.

[9] Padma Nyoman Crisnapati, I Nyoman Kusuma Wardana, and I Komang Agus Ady Aryanto. 2016. Rudas: Energy and sensor devices management system in home automation. In *2016 IEEE region 10 symposium (TENSYMP)*. IEEE, 184–187.

[10] Diego Corrêa da Silva, Denis Robson Dantas Boaventura, Mayki dos Santos Oliveira, Jander Pereira Santos Junior, Eduardo Ferreira da Silva, Eduardo Santana de Almeida, Cássio VS Prazeres, Ivan do Carmo Machado, Maycon Leone Maciel Peixoto, Gustavo Bittencourt Figueiredo, et al. 2025. Evaluating Multi-Label Machine Learning Models for Smart Home Environments. *Software: Practice and Experience* (2025).

[11] Mayki dos Santos Oliveira, Denis Robson Dantas Boaventura, Eduardo Ferreira Da Silva, Joel Machado Pires, Isaque Santana Copque, Bruno Pereira dos Santos, Ivan do Carmo Machado, Cássio Vinicius Serafim Prazeres, Maycon Leone Maciel Peixoto, Gustavo Bittencourt Figueiredo, et al. 2025. HESTIA: A Home Environment Simulator Targeting Inhabitant Activities. *Authorea Preprints* (2025).

[12] Stefan Fischer, Gabriela Karoline Michelon, Wesley KG Assunção, Rudolf Ramler, and Alexander Egyed. 2023. Designing a Test Model for a Configurable System: An Exploratory Study of Preprocessor Directives and Feature Toggles. In *Proceedings of the 17th International Working Conference on Variability Modelling of Software-Intensive Systems*. 31–39.

[13] Flagsmith. 2024. Flagsmith: Open-source Feature Flags and Remote Config. https://github.com/Flagsmith/flagsmith Accessed: 2024-12-11.

[14] Martin Fowler. [n. d.]. *Feature Flag*. https://martinfowler.com/bliki/FeatureFlag.html

[15] Hendrik Göttmann, Lars Luthmann, Malte Lochau, and Andy Schürr. 2020. Real-time-aware reconfiguration decisions for dynamic software product lines. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*. 1–11.

[16] GrowthBook. 2024. GrowthBook: Open-source Feature Flagging and Experimentation. https://github.com/growthbook/growthbook/ Accessed: 2024-12-11.
[17] Raj Jain. 1991. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons.
[18] Jean-Marc Jézéquel, Jörg Kienzle, and Mathieu Acher. 2022. From feature models to feature toggles in practice. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume A*. 234–244.
[19] Joselito Jr, Luana Martins, Dhyego Tavares, Denivan Campos, Frederico Durão, Cássio Prazeres, Maycon Peixoto, Gustavo Figueiredo, Ivan Machado, and Eduardo Almeida. 2024. Unleashing the Future of Smart Homes: A Revelation of Cutting-Edge Distributed Architecture. In *Anais do XVIII Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 41–50. https://doi.org/10.5753/sbcars.2024.3854
[20] LaunchDarkly. 2024. LaunchDarkly: Feature Flags as a Service. https://launchdarkly.com/ Accessed: 2024-12-11.
[21] Rezvan Mahdavi-Hezaveh, Nirav Ajmeri, and Laurie Williams. 2022. Feature toggles as code: Heuristics and metrics for structuring feature toggles. *Information and Software Technology* 145 (2022), 106813.
[22] Rezvan Mahdavi-Hezaveh, Jacob Dremann, and Laurie Williams. 2021. Software development with feature toggles: practices used by practitioners. *Empirical Software Engineering* 26 (2021), 1–33.
[23] Rezvan Mahdavi-Hezaveh, Sameeha Fatima, and Laurie Williams. 2024. Paving a Path for a Combined Family of Feature Toggle and Configuration Option Research. *ACM Trans. Softw. Eng. Methodol.* 33, 7, Article 172 (Sept. 2024), 27 pages. https://doi.org/10.1145/3672555
[24] Luana Martins, Denivan Campos, Joselito Mota, Dhyego Tavares, Jander Pereira, Mayki Oliveira, Denis Boaventura, Diego Correa, Eduardo Ferreira, George Pinto, et al. 2024. A Case Study of Smart Home Development. *IEEE Software* (2024).
[25] Ethan T McGee and John D McGregor. 2017. A realization effort estimation model for dynamic software product lines. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*. 111–116.
[26] Holt Mebane and Joni T Ohta. 2007. Dynamic complexity and the Owen firmware product line program. In *11th International Software Product Line Conference (SPLC 2007)*. IEEE, 212–222.
[27] Jens Meinicke, Juan Hoyos, Bogdan Vasilescu, and Christian Kästner. 2020. Capture the feature flag: Detecting feature flags in open-source. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 169–173.
[28] Jens Meinicke, Chu-Pan Wong, Bogdan Vasilescu, and Christian Kästner. 2020. Exploring differences and commonalities between feature flags and configuration options. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. 233–242.
[29] Bruno Iizuka Moritani and Jaejoon Lee. 2017. An approach for managing a distributed feature model to evolve self-adaptive dynamic software product lines. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*. 107–110.
[30] Juliana Alves Pereira, Sandro Schulze, Eduardo Figueiredo, and Gunter Saake. 2018. N-dimensional tensor factorization for self-configuration of software product lines at runtime. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*. 87–97.
[31] PostHog. 2024. PostHog: Product Analytics and Feature Flags. https://posthog.com/ Accessed: 2024-12-11.
[32] Md Tajmilur Rahman, Louis-Philippe Querel, Peter C Rigby, and Bram Adams. 2016. Feature toggles: practitioner practices and a case study. In *Proceedings of the 13th international conference on mining software repositories*. 201–211.
[33] Cosmin-Ioan Roşu and Mihai Togan. 2023. A Modern Paradigm for Effective Software Development: Feature Toggle Systems. In *2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE, 1–6.
[34] Split.io. 2024. Split.io: Feature Flags and Experimentation. https://www.split.io/ Accessed: 2024-12-11.
[35] Dhyego Tavares, Erlon P. Almeida, Jander P. S. Junior, Felipe S. A. Paixão, Enio G. Santana Jr., Rodrigo R. G. Souza, Frederico A. Durão, Cássio V. S. Prazeres, Ivan C. Machado, Gustavo B. Figueiredo, Maycon L. M. Peixoto, Hérsio M. Iwamoto, and Eduardo Santana de Almeida. 2024. Software Development Practices and Tools for University-Industry R&D Projects. In *XXIII Brazilian Symposium on Software Quality (SBQS 2024), November 5–8, 2024, Salvador, Brazil*. ACM, Salvador, Brazil. https://doi.org/10.1145/3701625.3701627 Proceedings of the XXIII Brazilian Symposium on Software Quality (SBQS 2024), November 5–8, 2024.
[36] Pablo Trinidad, Antonio Ruiz Cortés, Joaquin Pena, and David Benavides. 2007. Mapping Feature Models onto Component Models to Build Dynamic Software Product Lines.. In *SPLC (2)*. 51–56.
[37] Unleash. 2024. Unleash: Open-source feature management. https://github.com/Unleash/unleash/tree/main Accessed: 2024-12-11.
[38] Markus Weckesser, Roland Kluge, Martin Pfannemüller, Michael Matthé, Andy Schürr, and Christian Becker. 2018. Optimal reconfiguration of dynamic software product lines based on performance-influence models. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*. 98–109.
[39] Danny Weyns, M Usman Iftikhar, Sam Malek, and Jesper Andersson. 2012. Claims and supporting evidence for self-adaptive systems: A literature study. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 89–98.
[40] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 1–10.