Adaptive Software Development: A Comprehensive Framework Integrating Artificial Intelligence for Sustainable Evolution

Yaya Gadjama Soureya
Department of Mathematics and Computer Science, University of Douala, Cameroon
yayasoureya19@gmail.com

Ngoumou Amougou
Department of Mathematics and Computer Science, University of Yaounde, Cameroon
ngoumoua@yahoo.fr

Justin Moskolai Ngossaha
Department of Mathematics and Computer Science, University of Douala, Cameroon
moskojustin@gmail.com

Samuel Bowong Tsakou
Department of Mathematics and Computer Science
University of Douala, Cameroon
sbowong @gmail.com

Marcel Fouda Ndjodo
Department of Mathematics and Computer Science
University of Yaounde, Cameroon
foudandjodo @yahoo.fr

Abstract: *This research presents an innovative methodological framework for software development that integrates Artificial Intelligence (AI) techniques, Software Product Lines (SPL), and Lehman's [24] aging factors. The main objective is to improve the efficiency and adaptability of design processes for residential spaces through intelligent automation. This framework covers the entire software development life cycle, utilizing AI algorithms to optimize design and respond to the evolving needs of users while maximizing resource usage. A case study on a connected home concretely illustrates the application of this framework, demonstrating its effectiveness in creating dynamic and personalized designs. Furthermore, it addresses the issue of software sustainability by incorporating aging laws throughout their life cycle, an aspect often overlooked in existing solutions. By combining product line engineering and AI techniques, this framework offers a structured approach that promotes both sustainability and personalization. It has the potential to transform practices across various sectors, such as healthcare, finance, and education, while fostering a culture of sustainable innovation. However, its effectiveness also depends on the skills and experience of development teams, highlighting the importance of considering human factors in its application.*

Keywords: *Sustainability, evolution, quality of service, SPL, AI, IoT, connected homes.*

Received April 17, 2024; accepted January 03, 2025
https://doi.org/10.34028/iajit/22/2/4

1. Introduction

The software industry, characterized by intense competition [19] and an evolving nature, must meet user expectations while optimizing production resources in various contexts [8]. The main production challenges in this sector include reducing maintenance costs, improving productivity, and effectively managing competition [19]. An obvious solution to address these challenges is software durability. Indeed, effective control of software obsolescence ensures prolonged viability, which reduces production costs and enhances productivity [39]. According to Lehman [24], software durability is defined by its ability to endure over time and adapt to changes. This allows for reduced maintenance costs through proactive updates, thereby decreasing the time and resources required for upkeep [8]. Furthermore, improvements in productivity manifest through optimized performance and reduced downtime due to major failures, enabling users to remain productive. Moreover, paradigms such as Software Product Lines (SPL) facilitate the management of durability through concepts like component reuse, which reduces redundancy and development costs. They also allow for the application of updates across multiple products, thereby extending their lifespan [8], while offering flexibility and adaptability as well as efficient maintenance [39].

However, a crucial aspect of software is its intrinsic need to evolve, even without the requisite skills [6]. Maintaining existing systems proves significantly more costly, potentially reaching 2 to 100 times the cost of developing new systems [37]. This cost disparity often results from inadequate initial design. It is also important to remember that software is not produced merely for the sake of producing it, even if the production cost is low. The primary reason for the existence of software is user satisfaction. Moreover, the use of SPL requires an understanding of the software domain. Designing a domain is challenging, given the inability to fully grasp a domain thoroughly [30], especially in constantly evolving fields like the Internet of Things (IoT).

Even with a realistic domain design, software domains undergo continuous changes [1, 6, 9]. The complexities of a dynamic and evolving domain pose

challenges to achieving a sustainable and adaptable software design in this context. Therefore, designing a realistic software domain requires considering all current and future change factors.

In the field of software development methodologies, the incremental approaches proposed through the SPL paradigm aim to facilitate flexible evolution. However, these approaches often fail to explicitly address the factors of continuous change and their complex relationships [8]. The combination of the SPL paradigm with Artificial Intelligence (AI) promises to create customized software, improve coherence, and optimize SPL performance. Unfortunately, the vast field of AI lacks a guiding framework to effectively assist designers in its use. Existing literature primarily focuses on AI tools rather than techniques. Furthermore, the use of these tools is limited by mastery prerequisites and cost. Additionally, the effectiveness of AI tools depends on the initial quality of the product line construction.

This raises the question of how AI can be effectively used to achieve a realistic design of a dynamically coherent product line evolution, approaching the ideals of software eco-responsibility. To address this, we propose a methodological framework for the development of SPL [1] that integrates continuous change factors, as described by Lehman's [24] laws. The overarching goal is to create software capable of adapting seamlessly to continuous changes throughout its lifecycle while navigating the perpetual dilemma of user satisfaction versus production constraints. The aim, therefore, is ultimately to create software that is resilient over time while reducing production costs, but also ensuring customization according to the evolving preferences, skills, and needs of users.

The following section provides an overview of the considered context, highlighting its specificities as well as the complexity of the decision frameworks associated with it. A review of existing methods and approaches follows, leading to the presentation of the main objective of this article. A methodological framework is then proposed, accompanied by an illustrative example of the design of a connected home management system. Finally, a critical discussion addresses the capacity of this framework to account for other characteristics, particularly those specific to the design team in a software context, thereby leading to the conclusion and future perspectives of this study.

2. State of the Art

Several approaches exist to address the challenge of sustainability in software development by leveraging the similarities between software entities to generate new ones [37]. This contemporary manifestation of the copy-paste paradigm is embodied in methodologies such as SPL.

2.1. Software Product Line

In a given domain of activity, most software has similarities. Software Product Line Engineering (SPLE) accentuates this reuse of similarities [25]. A SPL is a paradigm advocating a modeling and development vision where the objective is not simply to obtain an individual software system but a set of software systems sharing common characteristics. This concept stems from the recognition that applications in various domains are not isolated systems but rather share common needs, functionalities and properties [32]. It is up to the development team to leverage these commonalities to define a fundamental architecture, simplifying the construction of new applications with increased efficiency and quality [38]. Indeed, SPL play several roles in software sustainability through code reuse, adaptability and scalability [8], contribution to sustainable development through reduction of software development costs, improvement of software quality as well as support for software sustainability [32]. This paradigm, which aims to systematize planned reuse throughout the software development process [34], is delimited into two main phases: domain engineering and application engineering, as illustrated in Figure 1.

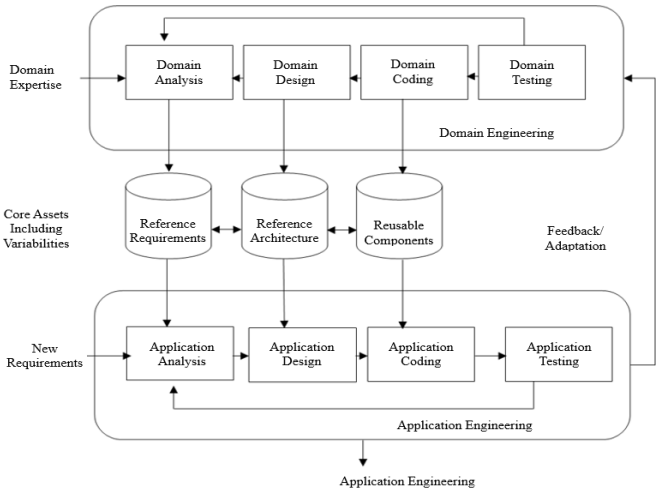


Figure 1. Product line development process [20].

The domain engineering phase, generally carried out by domain experts, consists of defining the basic elements that characterize the scope, their requirements, similarities and specificities as well as their relationships to define a basic architecture of the domain.

The application engineering phase consists of deriving an application from the basic architecture according to a defined context.

This paradigm, which appears as an ideal solution for software sustainability, nevertheless faces various challenges in fulfilling its mission, including the complexity of managing variability within product lines, maintaining the quality of variants, and managing reuse that can introduce complex dependencies and cause integration problems [8, 27]. Although challenges

exist in SPL, it is essential to mainly highlight their advantages in terms of software sustainability that can contribute to other areas. Indeed, by centralizing the management of variants from a common core, SPL enable efficient reuse of components, thus reducing development and maintenance efforts [8]. This approach also facilitates updates and patches, thus extending the software lifespan. By optimizing resources and minimizing maintenance efforts, SPL contribute to a reduction in the ecological footprint of software, thus promoting increased overall sustainability [5]. These advantages can be highlighted in the case of connected objects that evolve exponentially and where systematic and transversal reuse is emphasized [8]. Indeed, facing specific challenges such as rapid technological evolution, device obsolescence [6], frequent software updates, integration of new technologies [35], and changing user requirements, sustainability management becomes a key factor in reducing the maintenance costs of connected objects.

In addition to specific research on SPL, there are design methods that can be used. The Feature-Oriented Domain Analysis method (FODA) [30] employs techniques such as feature modeling, domain analysis, feature-oriented design, and variability modeling. Its main objective is to identify commonalities throughout the SPLE lifecycle and to effectively model variability and commonalities. However, FODA has several limitations. First, it lacks clear guidelines for managing dynamic complexity in evolving product line systems, making the models less useful in rapidly changing environments. Furthermore, as the number of features and their interactions increase, managing complexity becomes difficult, especially in product lines with many variants and dependencies [9].

Scalability is another issue; FODA can face challenges when it comes to large product lines, making the management of features and variants more complex, which can affect performance and documentation management. Furthermore, integrating the features defined by FODA with actual development tools can prove complex, requiring additional effort for validation and verification. Additionally, managing dependencies between features can lead to unforeseen effects on the system. Finally, FODA primarily focuses on variability at the initial design stage and may not effectively handle dynamic changes during the system's execution. As a result, it may not quickly adapt to rapid changes in requirements, necessitating significant manual adjustments to maintain feature consistency [8].

In this context, the Feature-Oriented Requirements Modeling method (FORM) [21] uses similar techniques, such as feature modeling, feature-oriented programming, domain analysis, and product line architecture. Although it is primarily designed for modeling variability during the requirements definition, design, and implementation phases, FORM has certain limitations. First, it does not comprehensively address all phases of the product lifecycle, which may require complementary approaches for complete management [21]. Additionally, modeling with FORM can become complex, especially for systems with many variations, leading to an increased need for time and specialized tools [11]. Regarding integration with other phases of software development, FORM may struggle to provide the detailed information needed, forcing teams to adopt additional approaches, complicating the process. Finally, like FODA, FORM does not provide clear guidelines for managing dynamic complexity in evolving systems, making the models less useful in rapidly changing environments.

On the other hand, the Function Analysis System Technique method (FAST) [4] focuses on feature modeling, domain analysis, configuration management, feature interaction analysis, and requirements analysis, playing a crucial role in product line testing. However, it also has certain limitations. First, the complexity of feature analysis can increase when managing a large number of features, complicating the analysis and design of products [4]. Additionally, while FAST is effective for managing features, it may lack the flexibility to quickly adapt to changes in market or customer requirements [4]. Finally, modeling dependencies between features can be challenging, especially when interactions become complex [4].

Continuing this exploration, the Software Product Line Integration and Testing method (SPLIT) [11] focuses on analyzing interactions between features, product configuration, constraint management, and variability management, thereby supporting variability management. However, it also presents some limitations. First, integrating different product variants can be complex, especially when interactions between components are complicated [11]. Furthermore, the testing process can become difficult to manage due to the diversity of possible configurations, which can lead to gaps in testing and undetected errors [11]. Finally, the resources required to manage integration and testing can be considerable [11].

As for the requiline method, it relies on a feature model based on FODA and aims to provide tools for modeling variability during the requirements definition phase. However, this method also has limitations. First, it heavily depends on requirements management and analysis, meaning that errors or inaccuracies in these can affect the quality of the product line [14]. Additionally, managing variants and specific configurations can become complex, especially in the presence of a large variety of products [14]. Finally, Requiline may struggle to adapt to environments where requirements frequently evolve [14].

Finally, to conclude, the Guendouz and Bennouar method [18] focuses on component modeling and variability during the architecture definition and product derivation phases. However, this method also has several limitations. First, the complexity of the models

used can create challenges when specifying different variants [18]. Additionally, implementing Guendouz and Bennouar [18] may require significant resources for training and process management. Lastly, the processes defined by Guendouz and Bennouar [18] may not be easily adaptable to the rapid changes in market requirements.

In general, these methods for analyzing and managing variability in SPL highlight the diversity of techniques used, such as FODA, FORM, and COVAMOF. While each method offers advantages, they share common limitations, including modeling complexity, high development costs, and insufficient adaptability to rapid requirement changes. Moreover, some methods primarily focus on static variability, which limits their effectiveness in the face of dynamic variations. Therefore, it is crucial to enhance the flexibility and integration of these approaches to better address the current challenges in software development.

In the context of connected objects, where requirements constantly evolve and variability can be particularly complex, it is essential to address these shortcomings to ensure effective management. Existing methods must be adapted and supported to meet the specific needs of this field. This ensures that the design and implementation of connected object systems are both agile and responsive to market fluctuations and user expectations.

2.2. Internet of Things (IoT)

The IoT, often regarded as the engine of sustainable development, is defined by Mohan and Ramesh [28] as a dynamic network infrastructure characterized by self-configuration, self-healing, and self-adaptation.

This infrastructure profoundly influences various sectors of human activity and represents a rapidly expanding industry facing challenges stemming from technological and material complexity [7, 19]. These challenges hinder comprehensive mastery [34] and contribute to the ongoing quest for cost reduction in production, deployment, and maintenance. Comprising data, algorithms, and an intelligent ecosystem, IoT has garnered significant research attention.

Garcia *et al*. [16] suggest viewing IoT as the internet of services, defining a service as a set of functions provided by software, typically accessible via an application or programming interface. Building upon this notion, Garcia *et al*. [16] propose a software development approach consisting of three stages: system operation description through graphical representation or source code, structural description, and code generation. However, this proposed approach fails to leverage similarities among connected software families and neglects factors such as obsolescence in design, as well as the essential properties of self-healing and self-adaptation inherent in intelligent systems.

Authors such as Krasner [23] endeavor to emphasize the significance of properties like self-healing and self-adaptation by introducing an approach that delineates variability through feature modeling. This approach defines, for each system condition, a resolution R to represent the set of triggered changes in terms of activation/deactivation of functionality, facilitating the deduction of product line architecture. However, the authors overlook factors related to domain evolution, and the definition of resolutions for any change remains limited due to the inability to anticipate all potential changes, given the incomplete control of the domain or user and potential future alterations. Consequently, there is no guarantee that the system will maintain consistency in the event of unforeseen changes.

In contrast, Tzeremes and Gomaa [37] leverage the similarities within software. Their work introduces an approach modeled on the SPL paradigm. Although they adopt the product line approach, the authors fail to consider factors that may influence the bank of product line elements and the parameters for customizing the application for individual users. Furthermore, there is no provision for ensuring self-healing, self-adaptation, or the efficacy of an intelligent system within their modeling framework.

However, AI can contribute to the key features that connected objects should have. Notably, self-repair, which is the ability of a system to detect and automatically correct errors or failures without human intervention. Self-adaptation, which is the capability of a system to modify its behaviors or parameters in response to changes in its environment without human intervention. The efficiency of intelligent systems, or the ability of a system to perform its tasks optimally in terms of resources and time. Indeed, AI systems can use machine learning techniques to continuously monitor systems and detect anomalies or abnormal behaviors that could indicate a failure, and apply automatic fixes or reconfiguration strategies to resolve the issue. They can also use learning algorithms to adjust their behaviors based on new data or environmental changes [6] and employ optimization techniques to manage resources more effectively, reducing costs and improving performance.

2.3. Artificial Intelligence and Software Product Line

The use of AI in SPL is generating increasing interest. Indeed, AI offers various essential functionalities, such as optimization, personalization based on the analysis of consumer behaviors and trends [38], as well as variability management.

For example, optimization helps improve the efficiency of development processes [34], while personalization assists in adapting products to the specific needs of users. Additionally, AI plays a crucial role in identifying inconsistencies within SPL. Through

predictive analysis, it can detect inconsistencies in models [5], study the system internally via its adaptation rules, and facilitate the derivation of new products.

Even better, we can combine the capabilities of AI with the benefits of SPL to optimize software design in the field of connected objects. The domain engineering phase, which involves understanding and modeling the requirements and characteristics of a specific domain, is divided into several key steps. First, requirements analysis can leverage Natural Language Processing (NLP) to examine specification documents, user feedback, and other sources to extract relevant requirements. The major challenge lies in the precise extraction of requirements from unstructured documents [34]. For example, for a connected home, an NLP model can analyze user reviews on various devices to identify the most requested features, such as energy management or security. Next, domain modeling using AI involves creating detailed and comprehensive ontologies that reflect the complexity of the domain [17]. In the context of a connected home system, this involves developing an ontology that includes concepts such as temperature control devices, motion sensors, and their interactions. Finally, process optimization via AI helps identify gaps between existing practices and best practices, while proposing process improvements without disrupting ongoing operations. For instance, using AI algorithms to analyze historical usage data from a connected home system can lead to suggestions for improvements in energy management and associated costs.

The application engineering phase focuses on the design, development, and deployment of software applications based on domain models. In design and architecture, AI is used to simulate different architectural configurations and optimize design choices in terms of performance and scalability [15]. The goal is to design a flexible architecture capable of seamlessly integrating various components. For example, in a connected home system, this involves developing a modular architecture where modules for lighting control, heating, and security are designed independently but integrated coherently. During development, AI tools enhance code quality through review processes [2], by validating code suggestions to ensure they are contextually appropriate and free of errors. AI-assisted code completion tools can, for instance, be used to develop scripts that automate temperature adjustments and lighting management in a connected home. Regarding testing and validation, AI is employed to create diverse and exhaustive test cases, with the objective of covering all potential usage scenarios. For a connected home system, this includes simulating events such as power outages or intrusions to verify the system's response. Finally, for deployment and maintenance, AI analyzes usage data to predict when updates or maintenance interventions will be needed, thereby reducing downtime and improving user satisfaction. For example, predictive maintenance can be implemented for connected home devices using AI to analyze usage patterns and schedule interventions before failures occur [6].

SPL play a crucial role in sustainable development by promoting reuse and standardization [16, 29]. AI can enhance the implementation of various phases in the SPL lifecycle. However, despite the benefits of combining AI with SPL, software aging remains inevitable. This means that while AI offers tools for managing and improving SPL, it must also be used to address the challenges posed by software aging, as outlined by Lehman's laws [24].

For instance, with laws like the Law of continuing change, which states that software must evolve to remain relevant, AI can assist by forecasting future needs and automating updates to handle this growing complexity. The Law of conservation of familiarity, which asserts that software retains its structure even when modified, can benefit from AI tools that recommend improvements to maintain code quality [13]. The Law of increasing complexity, which indicates that software becomes more complex over time, can be managed by AI through simplifying code structures and identifying redundancies. Additionally, for laws like the Adaptation to requirements, AI can facilitate this adaptation by automating the integration of new requirements and ensuring ongoing compliance [4].

Although AI offers promising solutions, there are still limitations in terms of a methodological framework for its use in SPL, particularly concerning the integration of aging laws. There is no unified methodological framework that systematically incorporates AI techniques into the management of SPL while considering the laws of aging. This limits designers' ability to apply these techniques consistently and effectively. Therefore, it is necessary to develop methodological frameworks that integrate AI practices with software aging management principles to provide clear guidelines and approaches tailored to the specific challenges encountered.

One of the challenges that remains in the design of software a line of product is the management of variability. Indeed, the identification, the representation of the variability and instantiation based on a specific product depending on the evolving context influenced among other things by user needs, market competitiveness, environmental constraints, user preferences and skills remain problematic.

These challenges can be addressed through AI, as shown in the literature, in the various phases of SPL.

The domain engineering phase. Can be broken down as follows:

In the need's identification stage, the goal is to define the domain and delineate its scope by collecting and analyzing information on customer expectations, functionalities, design, quality of service, as well as market trends, competitors, political constraints, user

skills, and preferences [7, 32]. At this stage, AI can play a key role by helping to extract and analyze requirements from various sources to ensure they align with user needs. Furthermore, the validation of requirements can also be facilitated by AI, which compares these requirements to user feedback and historical data to ensure they meet expectations [22].

Next, during the product family identification stage, AI's analysis of existing data allows for the identification of product line elements and their relationships, thus providing insights into commonalities and variations [30]. Additionally, AI algorithms can automate the identification of features by classifying them as common, variable, or optional based on user data and market analysis. Moreover, AI assists in modeling and visualizing relationships between product elements, thereby highlighting dependencies and constraints.

In the domain architecture generation stage, AI can also assist in modeling the elements of the product line by providing design recommendations based on existing patterns and best practices [22, 37]. Concurrently, it enables the identification of new product ideas and the evaluation of their potential in terms of demand, production costs, and profitability through the analysis of historical usage data [10].

Finally, for the domain compliance verification stage, AI can automate consistency testing by executing simulations and validations against established constraints, thereby ensuring compliance [23]. Moreover, AI supports ongoing compliance testing by monitoring changes in requirements and constraints, alerting teams to inconsistencies [25].

The application engineering phase:

In this context, the product derivation phase is divided into several key steps. First, during product specification, AI plays a crucial role by helping to generate and refine specifications based on user requirements [22]. Next, in product configuration, AI contributes by facilitating the selection of appropriate features from the product line. Furthermore, product customization is also enhanced by AI, which enables personalized adaptation based on user preferences and usage habits [17].

Regarding product consistency testing, AI can automate consistency checks, thus identifying discrepancies between product specifications and domain constraints [23]. Additionally, AI verifies that derived products conform to established rules and constraints within the domain. Finally, for product requirements validation, AI analyzes these requirements to ensure that derived products meet all specified criteria [7]. Thus, the integration of AI throughout these steps allows for the optimization of product design and derivation.

In the context of SPL, market engineering or market study is an essential phase that must be conducted separately [32], even though it is often integrated into domain engineering. This phase allows for the identification of specific user needs, evaluation of competition, and effective market segmentation. According to the literature, it provides crucial data to guide the design of software products, thereby ensuring their relevance and commercial success [7]. The integration of AI into this process facilitates market trend analysis and demand forecasting, making product development more agile and responsive [22].

The Market engineering phase: It consists of several essential phases that greatly benefit from the contributions of AI. First, in market analysis, identifying market needs is crucial; AI analyzes market data and trends to pinpoint user needs and preferences [7]. Concurrently, competitive analysis enables AI to evaluate competitors' offerings by examining their strengths and weaknesses [22, 32].

Next, during market segmentation, AI plays a decisive role by segmenting the market based on user demographics, behaviors, and preferences [26]. Additionally, it creates detailed buyer personas by analyzing user data and interactions, which helps refine marketing strategies [10].

Regarding market forecasting, AI utilizes historical data to predict future market demands, assisting in the effective allocation of resources [40]. Furthermore, trend analysis is facilitated by AI, which provides valuable insights into future market directions by examining current trends [26, 30]. Finally, in market strategy development, AI assists in formulating marketing strategies based on data-driven insights and forecasts, thereby optimizing strategic decisions [22, 32]. Thus, the integration of AI throughout these phases enhances the efficiency and relevance of market actions.

In summary, AI plays a key role in the different phases of engineering: it helps analyze and model requirements in domain engineering, facilitates the derivation of products tailored to market needs in application engineering, and enables trend analysis and demand forecasting in market engineering. However, domain, application, and market engineering in SPL represent complex processes where the literature often focuses on a specific Lehman's [24] law without providing a clear methodological framework that integrates all phases of software development. There is no systematic guide that highlights the Lehman's [24] laws, their relationships, and their placement in the process, nor does it specify how and where they manifest.

Although AI can be used to optimize these processes, its effectiveness would be significantly enhanced by a methodological framework that presents these laws. Indeed, this framework would serve as a guide to navigate through the various phases of software development, ensuring that all relevant Lehman's [24] laws are systematically and harmoniously integrated. This would help ensure the sustainability of the desired software as well as customer satisfaction.

The various AI techniques that can be used in the context of developing SPL to fulfill the previously mentioned roles are highlighted in Table 1. This table illustrates how these techniques contribute to the different phases of the development process.

Table 1. AI techniques used in product lines.

Product line phase	Challenges/Step	AI techniques	How AI addresses these challenges
Domain engineering	Identification of product line elements	Logic of description, ontological reasoning [8]	AI helps automate the analysis and modeling of requirements, making it easier to identify relevant elements
	Identifying inconsistencies	Description logic, knowledge-based reasoning [32]	AI algorithms can quickly identify and flag inconsistencies in requirements or models
	Managing variability	First-order logic, ontological reasoning [12]	AI assists in managing variability by providing reasoning capabilities to understand relationships between product features.
Application engineering	Product derivation	Ontological reasoning [7]	AI enhances the product derivation process by using ontologies to ensure that derived products meet specified requirements
	Product optimization	Optimization [22]	AI employs optimization techniques to improve product design based on user needs and market demands
	Test	Knowledge-based reasoning, genetic algorithms [32]	AI facilitates automated testing processes and uses genetic algorithms to explore potential solutions efficiently

The table summarizes the phases of SPL, highlighting the challenges, the AI techniques used, and their contributions.

In domain engineering, AI facilitates the identification of product line elements, detects inconsistencies, and helps manage variability. In application engineering, it enhances product derivation and optimizes design based on user needs and market requirements. Finally, AI automates testing through methods such as knowledge-based reasoning and genetic algorithms.

2.4. Main Goal and Positioning

In general, designers are consistently confronted with the dilemma of balancing user satisfaction against production constraints. Addressing this challenge necessitates prioritizing the resolution of production constraints before addressing user satisfaction. Enhancing production constraints entails achieving mastery over software durability, a task contingent upon a realistic understanding of the chosen domain. However, it is unrealistic to expect complete mastery over a professional field, regardless of one's expertise in the domain. Fields such as that of connected objects, for instance, are in a state of constant evolution, with technologies and needs inevitably evolving over time. Consequently, mastery over a domain remains elusive; even if a semblance of mastery is attained at a certain point, the laws of software obsolescence ensure that designs will eventually become outdated.

Existing approaches often treat software as isolated entities, neglecting the internal and external factors that have a significant impact on software sustainability. In addition, proposals generally focus on techniques rather than on the use of AI tools for the development process, which offer greater flexibility. Thus, the aim of this paper is to propose a methodological framework for the design of product lines considering the continuous integration of internal and external change factors, as dictated by software obsolescence laws, and the judicious use of AI techniques to improve service quality and efficiency.

3. Methodological Framework Proposed

Designing within a range remains a preferable option [18]. However, this design approach presents challenges, notably in managing the variability inherent in product ranges and adapting to internal and external factors that may alter the initial design. Consequently, the product line must evolve gradually over time to maintain control over variability while also ensuring adherence to Lehmann's [24] law of familiarity, which stipulates the necessity of maintaining familiarity over time for ease of user adoption [1]. We envision a process that maximizes consideration of factors influencing the product line to derive personalized and environmentally responsible applications, with the overarching goal of mastering software durability.

Achieving mastery over software durability entails a realistic understanding of the domain and adept management of changes. As evidenced by Lehman [24], software undergoes continuous internal and external aging processes, characterized by ongoing change, increasing complexity, preservation of familiarity, sustained growth, declining quality, and feedback from the system [1]. From these observations, several conclusions emerge: the inevitable decline in user satisfaction with software over time, the escalating complexity of software, the necessity for gradual evolution or change to facilitate user adoption, and the influence of market competitiveness and user demand for novelty, alongside changes in the software operating environment. Additionally, political constraints such as state requirements and societal shifts also impact software evolution. Thus, by defining evolution as a series of changes and transformations throughout the software lifecycle [6], we establish the context of a product line system at any given time as a combination of previously captured contexts, integrated with the current internal and external contexts.

A product line is subject to forces that can be broadly categorized into two major categories: internal forces and external forces.

Internal forces encompass everything that can influence the derivation of the application instance. This includes, among other things, the execution environment of the application, the specific needs of the user at a given moment, their skills, and their preferences.

External forces, on the other hand, represent everything that is not internal but can influence the final product. This includes user needs based on technological developments in the market, market competitiveness, and existing constraints.

In summary, internal forces are related to the characteristics and needs specific to the user and the application environment, while external forces concern the influences arising from market evolution and external conditions, as illustrated in Figure 2.

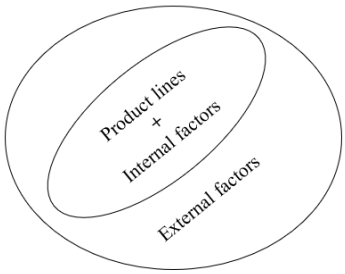


Figure 2. Product lines in an environment.

Laws and observations made, the dimensions according to which software is designed and evaluated are therefore, the needs users, user skills, quality of service (security, performance, ergonomics, addition of functionality, evolution of the market) sought after, technological and political developments.

The modeling of development processes is essential in the software domain, and several methods stand out for their effectiveness. First, Business Process Model and Notation (BPMN) allows for the creation of clear and understandable diagrams that illustrate the various processes involved in software development. Furthermore, BPMN plays a crucial role in managing variability by helping to model the processes that support these variations, enabling effective management of configurations and options. Its clear visual notation provides a detailed graphical representation of processes, thereby facilitating understanding and communication. Moreover, its international recognition promotes collaboration among international teams [33].

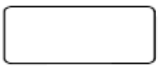
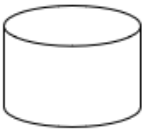
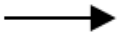
In addition, Unified Modeling Language (UML) focuses on technical modeling. It is used to create technical diagrams such as class diagrams, activity diagrams, and sequence diagrams in the context of SPL design [33]. Additionally, UML helps define the structure and interactions between software components [33]. However, this method can become complex, especially for large product lines with many components, and it is less suited for detailed business process modeling compared to BPMN.

Finally, Event-driven Process Chain (EPC) is well-suited for representing business processes in ERP systems, which is relevant for managing software development processes [18]. Nevertheless, EPC diagrams can be less visual and intuitive compared to those of BPMN, which can make them difficult to read, particularly for highly complex processes. Thus, although each method has its strengths and limitations, their appropriate use can greatly enhance the clarity and efficiency of software development processes.

In the context of managing SPL, BPMN stands out as the superior methodology compared to UML, and EPC. BPMN offers a clear and intuitive graphical representation of complex processes, making it easier to understand and manage the workflows and interactions involved. Its international standardization facilitates effective collaboration across diverse teams, while its comprehensive modeling capabilities support both high-level and detailed process documentation. Unlike UML, which is less focused on business process modeling, or EPC, which can be less visually intuitive and complex, BPMN provides a user-friendly approach that excels in optimizing process management and handling the variability within SPL.

For the reasons mentioned in the previous table, we will use the BPMN 2.0 method. The meanings of the different symbols are explained in Table 2:

Table 2. The different symbols used in BPMN.

Symbol	Signification
	Objects or entities
	Several documents
	and
	Activity or operation
	Data base
	Direction of flow

Taking into account the evolving factors over time, such as user needs, market competitiveness, constraints of the software domain, technology, user skills and preferences, as well as the environment in which the software is used, he proposed methodological framework is presented in Figure 3.

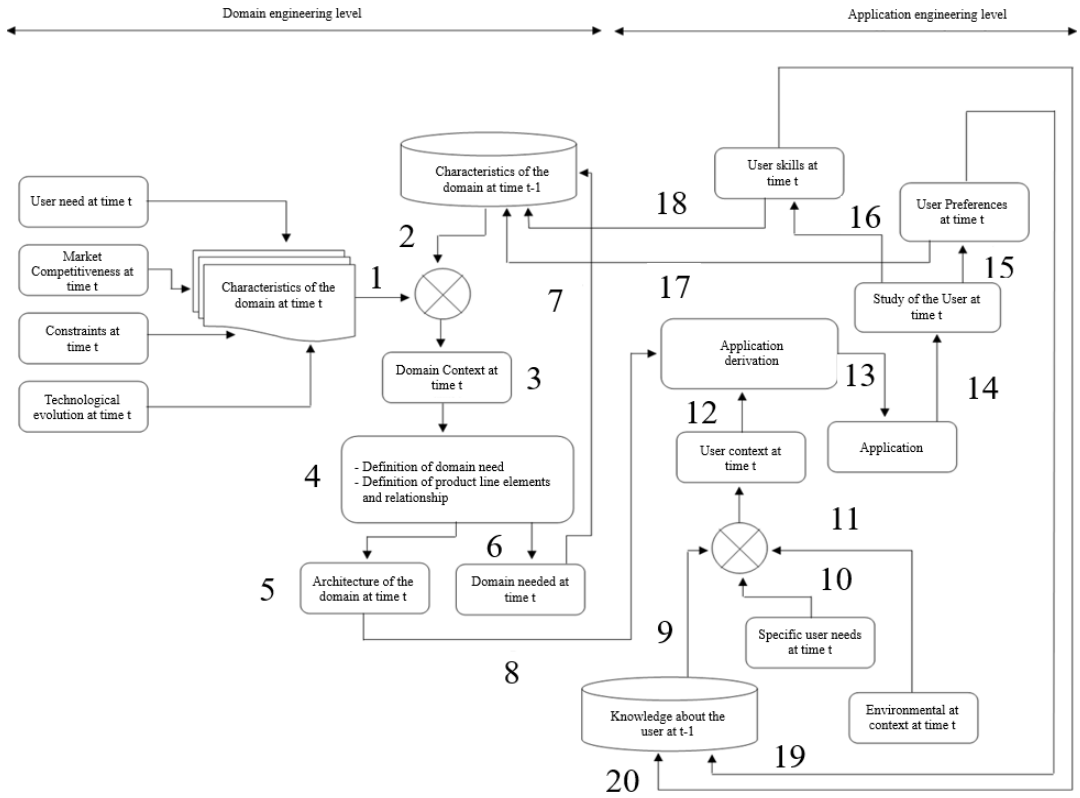


Figure 3. The proposed methodological framework for managing a product line.

It appears from Figure 3 that the design of the line of product and the derivation of the resulting products proceeds as follows:

- **Phase** 1 or domain engineering.
- *Step* 1: Define the characteristics of the domain at a given time t, by taking into consideration factors such as user needs, market competitiveness, existing constraints and technological developments at that time (1).
- *Step* 2: Define the context of the domain (3) in other words define the needs of the domain, the elements which constitute the SPL as well as their relationship (4) by taking into consideration the characteristics of the domain defined in step 1 plus the characteristics of the domain at time t-1 formerly known stored in database (2).
- *Step* 3: Derive from the domain context previously defined the architecture of the domain (5) and the real needs of the domain at time t (6).
- *Step* 4: Enrichment of the database of the context relating to characteristics for future use (7).
- **Phase** 2 or application engineering:
- *Step* 5: Define the real context of the user at time *t* (12) by taking into consideration the past knowledge that we have in the database on the user (9), plus the specific needs expressed by the user at time *t* (10) plus the environmental context of the user (11).
- *Step* 6: Derive the application corresponding (13) to the user context by considering the architecture of the domain obtained at the engineering phase of the domain (5) plus the real context of the user defined previously (12).
- *Step* 7: Study the user's data by analyzing the data from the application they use (14).
- *Step* 8: Determine the preferences of the user (15) at time t as well as his skills at time t (16) thanks to the study previously carried out (14).
- *Step* 9: Enrich the domain context database at the domain engineering phase (2) with data on user skills (18) and data on user preferences (17) for future derivation.
- *Step* 10: Enrich the database on the knowledge we have about the user (9) with the data obtained on skills (20) and those obtained on user preferences (19) for future derivation.

The previous framework takes into account that the environment defining the software domain and that of the user evolves over time. Thus, through an infinite loop system, where the system at time t is the system at the previous instant plus the additions at time t, the software derived from this methodological framework is virtually immune to aging over time.

4. Case Study

The choice of the type of use case can be justified by several factors. On one hand, connected objects appear as a promising solution for sustainable development in Africa, notably improving resource efficiency [31], facilitating access to services, and fostering innovation. On the other hand, their rapid scalability, increasing complexity in response to user needs, variability among

objects, and market competitiveness can be easily managed using the SPL paradigm [8].

Concerning the second question, to ensure robust and relevant results, comprehensive data collection is essential. In this case study, we aimed to fully understand user interactions and behaviors while ensuring that the design of the SPL aligns with real-world needs. Empirical data were collected, including direct observations and user feedback, which enabled us to gain insights into how users engage with the system. This understanding is crucial for fine-tuning features and functionality.

Such extensive data collection supports the application of Lehman [24] and AI laws, facilitating the system's evolution in response to user needs and technological changes.

Leveraging connected objects, this exhaustive data collection allows us to develop robust products that are relevant and adaptable to the changing demands of the market.

Regarding the proposed framework, the connected home will allow us to explore the design steps of a SPL. The aim will be to highlight the different instances of the home entities, which can be defined according to the environmental context.

Initially, in this article, we will explore the design steps of SPL from the proposed framework without considering Lehman's [24] vectors and without using AI techniques. Additionally, we consider that we are at the first iteration, meaning there is no previous iteration at t-1, which excludes later knowledge about the domain or the user.

In our case, we will design a functioning connected home for illustration purposes.

- **Phase** 1 or domain engineering phase. This phase consists of identifying the elements of the product line, their relationships, and the existing constraints.

According to the framework, we have:

- *Steps* 1 and 2: Definition of the domain.

Our domain or connected system consists of five entities: the multimedia entity that integrates the capabilities of multimedia devices; the presence simulation entity that gives the impression that there is at least one person; the presence detection entity that determines the presence of at least one person; the alarm entity that emits based on a given context; and the lighting entity that allows the light to turn on in the presence of someone. All these entities are interconnected. The consideration that we are at our first iteration allows us to proceed to Step 3.

- *Step* 3: Definition of the domain architecture and elements of the SPL.

For use case modeling, the featuring modeling method was used to enable clear graphical representation and facilitate comprehension. This method includes the following relationships: "Optional" means it can be selected or not, represented by a white circle; "Mandatory" is represented by a black circle and is called when the parent function is invoked; "Alternative" indicates several possibilities, represented by a black triangle; "Requires" represents a dependency relationship between two elements, associated with the keyword "Required."

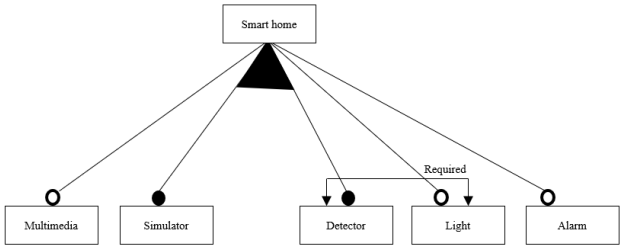


Figure 4. Connected home.

By applying this method to our project, the architecture of our connected home is outlined in Figure 4.

Table 3. Connected home truth table.

N^0	M	S	D	L	A
1.	0	0	0	0	0
2.	0	0	0	0	1
3.	0	0	0	1	0
4.	0	0	0	1	1
5.	0	0	1	0	0
6.	0	0	1	0	1
7.	0	0	1	1	0
8.	0	0	1	1	1
9.	0	1	0	0	0
10.	0	1	0	0	1
11.	0	1	0	1	0
12.	0	1	0	1	1
13.	0	1	1	0	0
14.	0	1	1	0	1
15.	0	1	1	1	0
16.	0	1	1	1	1
17.	1	0	0	0	0
18.	1	0	0	0	1
19.	1	0	0	1	0
20.	1	0	0	1	1
21.	1	0	1	0	0
22.	1	0	1	0	1
23.	1	0	1	1	0
24.	1	0	1	1	1
25.	1	1	0	0	0
26.	1	1	0	0	1
27.	1	1	0	1	0
28.	1	1	0	1	1
29.	1	1	1	0	0
30.	1	1	1	0	1
31.	1	1	1	1	0
32.	1	1	1	1	1

We can also identify the following operational constraints that govern the functioning of our system:

1. If at least one simulator is activated, then all presence detectors are deactivated.
2. If at least one detector is activated, then all presence simulators are deactivated.
3. Multimedia can be activated or not, independently of other entities.
4. No alarm is activated if the house is not empty.

5. If at least one alarm is activated, then at least one simulator is activated.
6. If the simulator is activated, then the alarm can be activated.
7. If the alarm and the simulator are activated at the same time, the system's operation is slowed down.

To better visualize the functioning of our system, let us note the states of the multimedia entities (M), the simulation (S), the detection (D), the lighting (L), and the alarm (A). If we denote M=1, this means that the entity M is activated, while M=0 indicates that the entity is not activated. The different cases representing the entities of the product lines of our connected home system can be deduced from the Table 3.

The different instances of the connected system, or actual elements of the product line that emerge while respecting the constraints, are lines 7, 10, 12, 23, 26, and 28. In other words:

- Multimedia is deactivated, simulator is disabled, detector presence is activated, the light is activated and the alarm is disabled.
- Multimedia is deactivated, the presence simulator is activated, the detector is deactivated, the light is deactivated and the alarm is activated.
- Multimedia is disabled, the simulator is disabled, the detector is deactivated, the light and the alarm are activated
- Multimedia is activated, the simulator is deactivated, the detector and light are activated and the alarm is deactivated.
- The multimedia is activated, the simulator is activated, the detector and the light are deactivated and the alarm is activated.
- Multimedia is activated, simulator is activated, detector is disabled, and the light and alarm are activated.

By using a standard notation such as modeling in descriptive language, the previous situation is represented as follows:

A concept corresponds to a class of elements and is interpreted as a set in a data universe:

Either
$CON=\{C1, C2, C3, C4, C5\}$ a finite set of concepts.

- $C1$: is the multimedia set.
- $C2$: is the simulator set.
- $C3$: is the set of detectors.
- $C4$: is the lamp set.
- $C5$: is the alarms set.

Either
$ROL=\{R1, R2\}$ a finite set of roles where:

- $R1=0$: is the non-activated state.
- $R2=1$: is the activated state.

We define the interpretation $\Delta^k ij:<C^k i, R_j>$ where Δ^k_i is the k-th element of the i th concept.

For $\Delta^k_{ij} \in <\Delta^k_i, R_j>$, $j \in \{1, 2\}$, Δ^k_{ij} is the k-th element of the I th concept activated or not. It means Δ^k_{i1} the k-th element of the i-th concept not activated and Δ^k_{i2} the k-th element of the i-th concept activated.

- N: set of natural numbers

Thus, we define the following operating constraints:

a) $\exists k \in N/\Delta k22 \Rightarrow \forall k' \in N, \Delta k'31$
b) $\exists k \in N/\Delta k32, \forall k' \in N$ we have $\Delta k'21$
c) $\exists k \in N/\Delta k1j \Rightarrow \forall k' \in N$ we have $\Delta k'ij$ with $i \neq 1, j \in \{1, 2\}$
d) $\exists k \in N/\Delta k32 \Rightarrow \forall k' \in N, \Delta k'51$
e) $\exists k \in N/\Delta k42 \Rightarrow \exists k' \in N, \Delta k'22$

- *Step* 4: In this step, the backup of the learned data about the domain at time t is automatic.

- **Phase** 2 or application engineering.

In our case, we consider that the user context is limited to the needs expressed at time t, without considering other influencing factors. For example, the need to activate a specific service.

- *Steps* 5 and 6: Definition of the context and derivation.

In the case of a smart home like ours, we assume that the multimedia and lighting functionalities are manual, while the functionalities of the simulator, presence detector, and alarm are automatic, as the system can determine whether there is a person in the house or not. For example, we can say that the user wants to activate the multimedia functions and that the light should turn on if there is no one in the room.

In the scenario where a user is absent from their home and has left the multimedia and lights on, the instance that most closely matches this situation is the sixth: the multimedia is activated, the simulator is activated, the detector is disabled, and both the light and alarm are also activated.

5. Discussion

This case study provides valuable insights into the application of the proposed methodological framework for designing a line of software products that integrate Lehman's laws [24]. Indeed, this framework establishes a solid foundation for structured software development, combining sustainability and customization. It addresses the challenge of reconciling these two aspects by ensuring that the developed software is not only sustainable but also tailored to the specific needs of users.

Moreover, the continuous approach to software development, which incorporates feedback at each iteration, is essential for maintaining the relevance and effectiveness of solutions over time. Due to its intuitive and easy-to-follow nature, this framework is ideal for designing software that requires adaptability, scalability,

facilitation of innovation, and continuous improvement, while also serving as a starting point for deeper analyses in real-world contexts.

The software developed according to this framework demonstrates resistance to obsolescence, thanks to a system that continuously considers factors influencing its evolution. However, we argue that this framework alone does not guarantee the expected success, namely, near-infinite sustainability coupled with effective customization. It is essential to identify and apply appropriate engineering methodologies for the recognition, interpretation, modeling, and implementation of software design elements.

Furthermore, even with the use of innovative AI tools, it is crucial to carefully select the data to be integrated from the outset, as this choice significantly influences the smooth progression of the entire design process. Thus, the skills and experience of the team are paramount in determining the inputs to be provided to the relevant engineering departments. The characteristics of the team, the type of project, and other factors should not be overlooked in the pursuit of the desired research objective, as illustrated in Table 4.

Table 4. Factors that can influence the success of software development [36].

Influence factors	frequency
Team capabilities and experience	64
Programming language experience	16
Application experience and familiarity	16
Project manager experience and skills	15
Software complexity	42
Database size and complexity	9
Architecture complexity	9
Complexity of interface to other systems	8
Project constraints	41
Schedule pressure	43
Decentralized/multisite development	9
Tool usage and quality/effectiveness	41
Case tools	12
Testing tools	5
Programming language	29
Domain	14
Development type	11

Indeed, the experience of the team can represent an influence ranging from 42% to 60.5%, while the choice of approach can potentially have an influence of up to 80%. This underscores the importance of a well-established methodological framework developed with the support of a competent and experienced team to ensure the success of a software design project. The elements influencing the dimensions of constraints and competitiveness depend on the specific objectives of the project as well as the perceptions of these dimensions. However, identifying the appropriate elements for these dimensions remains a delicate and complex task.

6. Conclusions and Future Directions

The objective of this research was to propose a solution to the problem of software sustainability by establishing a methodological framework that considers the aging laws of software throughout its life cycle, in conjunction with product line engineering and AI techniques. The literature review revealed that existing solutions did not comprehensively consider Lehman's [24] aging factors, nor did they do so continuously. In response, we developed a framework that provides a solid foundation for structured software development, combining sustainability and customization. The case study on the design of a smart home demonstrated that the developed software can be both sustainable and tailored to the specific needs of users.

However, the application of this framework must be approached with caution, as subjective factors, such as the experience and skills of the development team, can influence its effectiveness. We emphasize the importance of not neglecting human skills and recommend integrating appropriate AI techniques while considering Lehman's [24] factors, such as market competitiveness and user skills.

To evaluate the effectiveness of our framework once implemented, we propose using several metrics, including software lifespan, user satisfaction, and maintenance costs. Moreover, the adoption of feedback loops will ensure continuous improvement in response to user needs.

In the future, software sustainability will become a crucial issue considering the rise of digital technologies and growing environmental concerns. Our methodological framework can play a key role by integrating sustainable development practices from the earliest stages. By facilitating the creation of adaptable and environmentally friendly solutions, it can help companies reduce their carbon footprint and meet the increasing expectations for sustainability.

In conclusion, this methodological framework has the potential to transform practices in the software industry and could bring significant benefits to various sectors such as information technology, healthcare, finance, manufacturing, and education. It represents a unique opportunity to rethink software development to ensure both efficiency and sustainability while fostering a culture of sustainable innovation.

As part of our ongoing research, we will focus on intelligent forecasting systems using the dynamic systems design approach we have proposed, and we will strive to address the challenges related to [3].

References

[1] Abo Zaid L., Kleinermann F., and De Troyer O., "Applying Semantic Web Technology to Feature Modeling," *in Proceedings of the ACM Symposium on Applied Computing*, Hawaii, pp. 1252-1256, 2009. https://dl.acm.org/doi/10.1145/1529282.1529563

[2] Ajiga D., Okeleke P., Folorunsho S., and Ezeigweneme C., "Enhancing Software Development Practices with AI Insights in High-

Tech Companies," *Computer Science and IT Research Journal*, no. 5, vol. 8, pp. 1897-1919, 2024. https://doi.org/10.51594/csitrj.v5i8.1450

[3] Al Mokhtar Z. and Dawwd S., "3D VAE Video Prediction Model with Kullback Leibler Loss Enhancement," *The International Arab Journal of Information Technology*, vol. 21, no. 5, pp. 879-888, 2024. DOI:10.34028/iajit/21/5/9

[4] Apel S., Batory D., Kastner C., and Saake G., *Feature-Oriented Software Product Lines: Concepts and Implementation*, Springer, 2013. https://doi.org/10.1007/978-3-642-37521-7

[5] Bogart C., Kastner C., Herbsleb J., and Thung F., "When and How to Make Breaking Changes: Policies and Practices in 18 Open Source Software Ecosystems," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 4, pp. 1-56, 2021. https://doi.org/10.1145/3447245

[6] Carneiro D., Guimaraes M., Silva F., and Novais P., "A Predictive and User-Centric Approach to Machine Learning in Data Streaming Scenarios," *Neurocomputing*, vol. 484, pp. 238-249, 2022. https://doi.org/10.1016/j.neucom.2021.07.100

[7] Cheruvu S., Kumar A., Smith N., and Wheeler D., *In Demystifying Internet of Things Security: Successful IoT Device/Edge and Platform Security Deployment*, Apress, 2020. https://doi.org/10.1007/978-1-4842-2896-8_2

[8] Clements P. and Northrop L., *Software Product Lines: Practices and Models*, Addison-Wesley Professional, 2001. https://dl.acm.org/doi/10.5555/501065

[9] Clements P., *Software Product Lines: Practices and Patterns*, Addison-Wesley Longman Publishing, 2001. https://dl.acm.org/doi/10.5555/501065

[10] Cooper R., "The Drivers of Success in New-Product Development," *Industrial Marketing Management*, vol. 76, pp. 36-47, 2019. https://doi.org/10.1016/j.indmarman.2018.07.005

[11] Coriat M., Jourdan J., and Boisbourdin F., *Software Product Lines Experience and Research Directions*, Springer, 2000. https://doi.org/10.1007/978-1-4615-4339-8_8

[12] Crouse M., Abdelaziz I., Makni B., Whitehead S., Cornelio C., Kapanipathi P., Srinivas K., Thost V., Witbrock M., and Fokoue A., "A Deep Reinforcement Learning Approach to First-Order Logic Theorem Proving," *AAAI Technical Track on Knowledge Representation and Reasoning*, vol. 35, no. 7, pp. 6279-6287, 2021. https://doi.org/10.1609/aaai.v35i7.16780

[13] Cunningham W., "The WyCash Portfolio Management System," *ACM SIGPLAN OOPS Messenger*, vol. 4, no. 2, pp. 29-30, 1992. https://doi.org/10.1145/157710.157715

[14] De Oliveira R., Insfran E., Abrahao S., Gonzalez-Huerta J., Blanes D., and Cohen S., "A Feature-Driven Requirements Engineering Approach for Software Product Lines," *in Proceedings of the 7th Brazilian Symposium on Software Components, Architectures and Reuse*, Brasilia, pp. 1-10, 2013. https://ieeexplore.ieee.org/document/6685785

[15] Feldt R., De Oliveira Neto F., and Torkar R., "Ways of Applying Artificial Intelligence in Software Engineering," *in Proceedings of the 40th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Gothenburg, pp. 35-41, 2018. https://doi.org/10.1145/3194104.3194109

[16] Garcia S., Struber D., Brugali D., Di Fava A., Pelliccione P., and Berger T., "Software Variability in Service Robotics," *Empirical Software Engineering*, vol. 28, no. 24, pp. 1-67, 2023. https://link.springer.com/article/10.1007/s10664-022-10231-5

[17] Gruber T., "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220, 1993. https://doi.org/10.1006/knac.1993.1008

[18] Guendouz A. and Bennouar D., "Component-Based Specification of Software Product Line Architecture," *in Proceedings of the 1st International Conference on Advanced Aspects of Software Engineering*, Constantine, pp. 100-107, 2014. https://ceur-ws.org/Vol-1294/paper11.pdf

[19] Hussein A., "Internet of Things (IoT): Research Challenges and Future Applications," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 6, pp. 77-82, 2019. https://doi.org/10.14569/IJACSA.2019.0100611

[20] Jean-Christophe., "Software Product Lines: Reuse and Variability," *Smals Techno 35*, pp. 1-16, 2009. https://www.smals.be/sites/default/files/assets/techno35-fr.pdf

[21] Kang K., Kim S., Lee J., Kim K., Shin E., and Huh M., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering*, vol. 5, no. 1, pp. 143-168, 1998. https://doi.org/10.1023/A:1018980625587

[22] Kapferer S. and Zimmermann O., *Model-Driven Engineering and Software Development*, Springer, 2021. https://doi.org/10.1007/978-3-030-67445-8_11

[23] Krasner H., "The Cost of Poor Software Quality in the US: A 2020 Report," *Consortium for Information and Software Quality*, pp. 1-46, 2021. https://www.it-cisq.org/cisq-files/pdf/CPSQ-2020-report.pdf

[24] Lehman M., "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060-1076, 1980. https://doi.org/10.1109/PROC.1980.11805

[25] Linden F., Schmid K., and Rommes E., *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*, Springer, 2007. https://doi.org/10.1007/978-3-540-71437-8

[26] Melo A., Fagundes R., Lenarduzzi V., and Santos W., "Identification and Measurement of Requirements Technical Debt in Software Development: A Systematic Literature Review," *Journal of Systems and Software,* vol. 194, pp. 111483, 2022. https://doi.org/10.1016/j.jss.2022.111483

[27] Metzger A. and Pohl K., "Variability Management in Software Product Line Engineering," *in Proceedings of the 29th International Conference on Software Engineering*, Minneapolis, pp. 186-187, 2007. https://doi.org/10.1109/ICSECOMPANION.2007.83

[28] Mohan K. and Ramesh B., "Ontology-Based Support for Variability Management in Product and Service," *in Proceedings of the 36th Annual International Conference on System Sciences*, Hawaii, pp. 1-10, 2003. https://doi.org/10.1109/HICSS.2003.1174190

[29] Naumann S., Kern E., Dick M., and Johann T., *ICT Innovations for Sustainability*, Springer, 2014. https://doi.org/10.1007/978-3-319-09228-7_11

[30] Niemela E., "Strategies of Product Family Architecture Development," *in Proceedings of the 8th International Conference on Software Reuse*, Rennes, pp 186-197, 2005. https://doi.org/10.1007/11554844_21

[31] Ooko S., Ogore M., Nsenga J., and Zennaro M., "TinyML in Africa: Opportunities and Challenges," *in Proceedings of the IEEE Globecom Workshops*, Madrid, pp. 1-6, 2021. https://doi.org/10.1109/gcwkshps52748.2021.9682107

[32] Pohl K., Bockle G., and Van der Linden F., *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer, 2005. https://doi.org/10.1007/3-540-28901-1

[33] Rumbaugh J., Blaha M., Lorensen W., Eddy F., and Premerlani W., *Object-Oriented Modeling and Design*, Prentice Hall, 1991. https://dl.acm.org/doi/10.5555/130437

[34] Sanalqah H., Kang S., and Lee J., "A Method to Optimize the Scope of a Software Product Platform Based on End-User Features," *Journal of Systems and Software*, vol. 98, pp. 79-106, 2014. https://doi.org/10.1016/j.jss.2014.08.034

[35] Solomon B., Maynard M., and Khomh F., "Maintenance Cost of Software Ecosystem Updates," *in Proceedings of the 6th International Conference on Emerging Data and Industry EDI40*, Leuven, pp. 608-615, 2023. https://doi.org/10.1016/j.procs.2023.03.077

[36] Trendowicz A. and Munch J., "Factors Influencing Software Development Productivity: State-of-the-Art and Industrial Experiences," *Advances in Computers*, vol. 77, pp. 185-241, 2009. https://doi.org/10.1016/S0065-2458(09)01206-6

[37] Tzeremes V. and Gomaa H., "A Software Product Line Approach to Designing End User Applications for the Internet of Things," *in Proceedings of the 13th International Conference on Software Technologies*, Porto, pp. 656-663, 2018. https://www.scitepress.org/papers/2018/69049/69049.pdf

[38] Wolfert B., Ge L., Verdouw C., and Bogaardt M., "Big Data in Smart Farming-A Review," *Agricultural Systems*, vol. 153, pp. 69-80, 2017. https://doi.org/10.1016/j.agsy.2017.01.023

[39] Yli-Huumo J., Maglyas A., and Smolander K., *Product-Focused Software Process Improvement*, Springer, 2014. https://doi.org/10.1007/978-3-319-13835-0_7

[40] Zulkarnain. and Putri T., "Intelligent Transportation Systems (ITS): A Systematic Review Using a Natural Language Processing (NLP) Approach," *Heliyon*, vol. 7, no. 12, pp. 1-15, 2021. https://doi.org/10.1016/j.heliyon.2021.e08615

Yaya Gadjama Soureya is a Ph.D. student at the University of Douala, Cameroon. Originally from the Far North Region of Cameroon. She holds a research Master's Degree in Computer Science from the University of Yaoundé I, Cameroon. Her main research interests include Software Product Lines, Software Cost Estimation, Artificial Intelligence and the Internet of Things.

Ngoumou Amougou is a Native of Center Region Cameoon. He has received a Ph.D. in Computer Science at the University of Yaounde I, Cameroon and currently works as a Senior Lecturer of Computer Science at the Department of Computer Science of the Higher Teacher Training College of the University of Yaounde I, Cameroon. His main research interests include Software Product Lines, Software Cost Estimation, Domain-Specific Languages and Information Systems.

Justin Moskolai Ngossaha is a Senior Lecturer and Researcher specializing in Computer Science. He is currently affiliated with the Department of Mathematics and Computer Science at the University of Douala. His research focuses on Urban Data Management and Artificial Intelligence, particularly in the Context of Sustainable Complex Systems. He has made significant contributions to various research projects and publications in these fields, emphasizing the development and application of Decision Support Tools to Enhance the Design and Implementation of Sustainable Solutions.

Samuel Bowong Tsakou obtained his Doctorate in Applied Mathematics at the University of Metz, under the supervision of Gauthier Sallet entitled Contribution to the Stabilization and Stability of Nonlinear Systems. He is Professor and Head of Mathematics and Computer Science Department at the Sciences Faculty of the Douala University, Cameroon. Head of the "Modelling, Analysis and Simulation in Epidemiology and Immunology" team (GRIMCAPE). Member of the EPIdemiological modeling and control for Tropical Agriculture (EPITAG) team. Member of the Unit for Mathematical and Computer Modeling of Complex Systems (UMISCO).

Marcel Fouda Ndjodo is a native of Center Region Cameroon. He has received a Ph.D. in Computer Science at the University of Aix-Marseille II, France and currently works as a full Professor of Computer Science and the Head of the Computer Science Department of the Higher Teacher Training College of the University of Yaounde I, Cameroon. He coordinates besides the information systems and numerical technologies of education at the higher teacher training college. He has authored of many scientific publications and has supervised many Ph.D. Thesis in Information Systems and Software Engineering.