Journal of Posthumanism
2026
Volume: 6, No: 2, pp. 671–688
ISSN: 2634-3576 (Print) | ISSN 2634-3584 (Online)
posthumanism.co.uk

DOI: https://doi.org/10.63332/joph.v6i2.4014

Development of Software Assets Based on Reuse

Erika Orbes[1], Alexander Barón Salazar[2], Oscar Revelo Sánchez[3]

Abstract

The objective of this research was to propose an essentialized practice for the development of software assets based on reuse, in order to improve its understanding and application in software development contexts. The study addresses the limitations of traditional textual descriptions of practices, which are often ambiguous or difficult to interpret. To overcome this issue, the Model for Unified Definition of Practices in Software Engineering was applied to identify and structure the essential elements of the proposed practice, ensuring clarity and proper naming. The methodology included a review of conceptual references and a theoretical study on software asset reuse, followed by the construction of the practice using the essential practices approach. The practice was validated through expert consultation with Software Engineering professionals, who evaluated its clarity, usefulness, and applicability. The results show that essentialization supports clearer definition and effective application of the practice.

Keywords: *Software Reuse, Software Assets, Essentialization, Software Engineering, Essence Kernel*

Introduction

Development of Software Assets based on Reuse

The application of best practices in software engineering helps reduce costs, effort, and time, while maintaining the integrity of the final product. A set of practices forms a methodology, and through the adoption of various development methodologies, teams can generate solutions that reflect consistent project progress. However, many software engineering practices are defined in different ways, which creates difficulties in understanding and implementation. One such case is the development of software assets based on reuse, which involves leveraging existing assets to address different problems. In this context, commonly reused assets include libraries, modules, components, source code, and commercially available products. Software development based on reuse is widely recognized as a systematic and effective practice that follows a structured and repeatable process. It has the potential to significantly improve productivity, quality, and cost-efficiency, which has made it increasingly relevant for development teams.

Despite its advantages, this practice is often described in textual formats that hinder its application, particularly when trying to identify components such as activities, tasks, steps, input and output criteria, and work products. To address this challenge, there are strategies that aim to improve the understanding and application of engineering practices. One such strategy is essentialization, which focuses on identifying the necessary and sufficient elements to define a practice effectively. This article presents the essentialization of the practice of developing software assets based on reuse through the application of the Model for Unified Definition of

[1] Universidad de Nariño, Colombia; Email: erika.ax.18@gmail.com
[2] Universidad de Nariño, Colombia; Email: abaron_98@udenar.edu.co
[3] Universidad de Nariño, Colombia; Email: orevelo@udenar.edu.co

Practices in Software Engineering. By using this model, essential elements are identified and structured to define a well-formed, well-named, and easy-to-understand and apply practice. A practice is considered essentialized when it is coherent, consistent, sufficient, and properly named. To demonstrate that the practice has been correctly essentialized, a validation process was carried out through expert evaluation using the focus group technique, with participants selected based on predefined expertise criteria.

This article is an extended version of the work titled "Systematic Reuse of Software Assets: An Essentialized Practice in Software Engineering", originally presented at the 7° Congreso Internacional de Ingenierías 2024 (Orbes et al., 2024). The main extensions include a change in the practice's name, adopted after the expert evaluation process. Additionally, the section on conceptual foundations was strengthened, and the study on the development of software assets based on reuse was expanded. A new section on contributions and opportunities for improvement was also added, along with additional bibliographic references to enrich the theoretical discussion.

Methodology

This work was carried out in four phases, based on the methodology developed by (Barón, 2019), as described below:

Table 1: Methodology.

Phase	Activities
Exploration	Consultation of conceptual references
Analysis	Study on the development of software assets based on reuse
Construction	Essentialization of the Development of Software Assets Based on Reuse Practice
Validation	Expert evaluation of the essentialized practice

Similarly, for the construction phase, the model for defining practices in Software Engineering proposed by (Barón, 2019), is applied.

Results and Discussion

Exploration Phase - Consultation of Conceptual References

Development of software assets based on reuse. Software reuse is a process that leverages existing resources to solve various problems. According to Nikolaidis et al. (2024), software reuse involves using available solutions to build new software or improve existing systems, avoiding development from scratch. In software development, the resources commonly reused are not limited to source code; software assets also include libraries, modules, components, among others. For example, Guo et al. (2024), presents the reuse of unit test cases. Similarly, Adjandra et al. (2021), discusses the reuse of knowledge in software development. The benefits of developing software through reuse include improvements in productivity, quality, and cost reduction (IEEE Computer Society, 2004). According to Bibi et al. (2023), reuse is gaining popularity due to its potential to save developers time and effort.

Software reuse encompasses two closely related practices: "building for reuse" and "building with reuse." The former involves creating reusable software resources, while the latter focuses on reusing existing software resources to construct a new solution. The latter practice is the focus

of study in this work (IEEE Computer Society, 2004).

Essence for Practice Definition. The Software Engineering Method and Theory (Semat) community has consolidated a framework called Essence to restructure software engineering and improve its methods. The standard defines methods as a set of practices that can be used to represent the activities carried out during the development process. In Essence, a practice is defined as a way of addressing an activity that can be repetitive, fulfilling a specific purpose. Moreover, a practice can be reused by any method (Object Management Group, 2014).

SEMAT promotes a scalable and viable standard that facilitates the transfer of software engineering methods and practices, adapting, measuring, and comparing them (Zapata Jaramillo & Henao Roqueme, 2021).

Essence consists of a core and a language. The core is a set of concepts and their relationships. These concepts and relationships are essential and always present in any software development process. The language is a set of elements and expression rules for representing the development process (Gómez Álvarez et al., 2018).

Semat establishes a common ground for software engineering through a kernel of essential elements applied generally across all projects. The kernel includes "things we always work with," "things we always do," and "necessary skills" when developing a software system (Jacobson et al., 2012).

Things you always work with. Represented as alphas that enable evaluation of progress and health in software engineering efforts. Alphas consist of a set of states, and each state has a checklist of criteria required for its fulfillment. Alphas make the kernel actionable and applicable in any practice, regardless of specifics. The alphas defined in the kernel are shown in Figure 1 (Object Management Group, 2014).

Things you always do. Represented as a set of activity spaces that complement alphas, meaning they lead to state changes in an alpha. Figure 2 presents the activity spaces proposed by the kernel (Object Management Group, 2014).

Necessary skills. Competencies complement the alphas and activity spaces, establishing key capabilities required for software engineering efforts. Figure 3 presents the competencies defined in the kernel (Object Management Group, 2014).

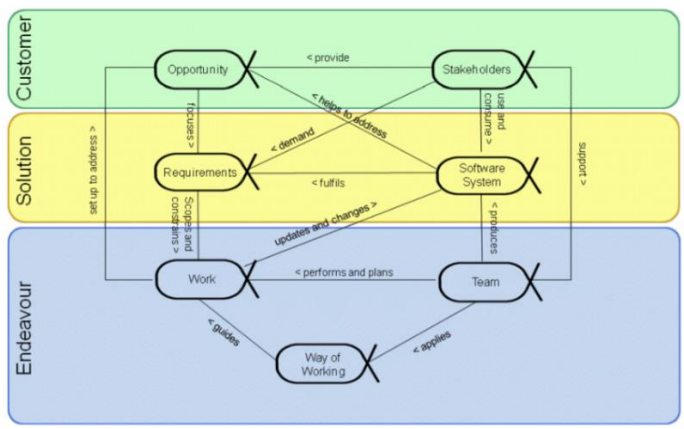

Figure 1: The Kernel Alphas.

Note. Reproduced from Kernel and language for software engineering methods (Essence) (Version 1.0), by Object Management Group (2014), https://www.omg.org/spec/Essence/1.0/

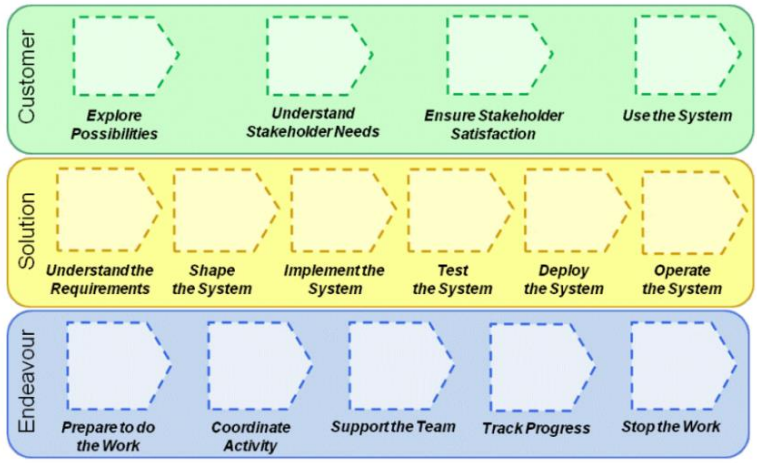

Figure 1. Activity Spaces.

Note. Reproduced from Kernel and language for software engineering methods (Essence) (Version 1.0), by Object Management Group (2014), https://www.omg.org/spec/Essence/1.0/

Figure 2. Competencies

Note. Reproduced from Kernel and language for software engineering methods (Essence) (Version 1.0), by Object Management Group (2014), https://www.omg.org/spec/Essence/1.0/

Model for Defining Software Engineering Practices. The model integrates a series of systematically structured components to fulfill specific functions that guide the definition of well-formed and well-named practices. Figure 4 illustrates the components of the model (Barón, 2019).

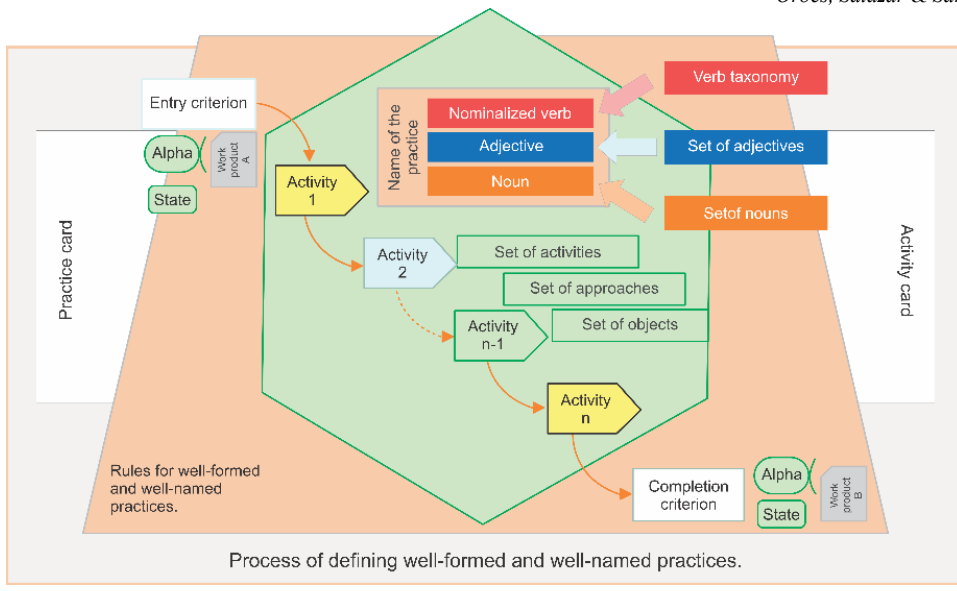

Figure 3. Components of the Model.

Note. Reproduced from Modelo para la definición unificada de la práctica como constructo teórico en ingeniería de software (doctoral dissertation), by Baron, A. (2019), Universidad Nacional de Colombia, Institutional Repository UNAL. https://repositorio.unal.edu.co/handle/unal/76404

A practice is well-formed if it adheres to the rules of coherence, consistency, and sufficiency (Barón, 2019).

Coherence Rule. A practice is coherent if the completion criteria for each activity contribute to the progression toward the practice's completion criteria (Barón, 2019).

Consistency Rule. The activities of a practice are consistent if their completion criteria align with the input criteria of another activity, and at least one activity's input criteria matches the practice's input criteria, similarly for the output criteria (Barón, 2019).

Sufficiency Rule. The activities of a practice are sufficient if the completion criteria for each activity lead to meeting the practice's completion criteria (Barón, 2019).

A practice is well-named if it follows the rules for naming a practice, which guide the definition of the nominalized verb, adjective, and noun that make up the practice name (Barón, 2019).

Practice and Activity Cards. Cards facilitate visualization of essential elements of the practice and its activities, aiding in the management of the practice's application. Figures 5 and 6 present examples of practice and activity cards (Barón, 2019).

Software engineering practice **Name:** [Nominalized verb] [Adjective][Noun] **Description:** [Brief description of the practice]
Entry Criterion
[Partially in:] [(Noun: State)]
Work products associated with the entry criterion **1.** [List the work products that serve as evidence of meeting the entry criterion] **2.** ...
Completion Criterion
[Contributes to:] [(Noun: State)]
Work products associated with the completion criterion **1.** [List the work products that are evidence of meeting the completion criterion] **2.** ...

Figure 4. Practice Card.

Note. Reproduced from Modelo para la definición unificada de la práctica como constructo teórico en ingeniería de software (doctoral dissertation), by Baron, A. (2019), Universidad Nacional de Colombia, Institutional Repository UNAL. https://repositorio.unal.edu.co/handle/unal/76404

Activity [#]: [Activity Name] **Approach:** [activity approach] **Activity Space:** [Activity space that groups the activity] **Description:** [Brief description of the activity]
Entry Criterion
[(Noun : State) or (Work product : Level of detail)]
Work products associated with the entry criterion **1.** [List the work products that are evidence of meeting the entry criterion] **2.** ...
Tasks
1. [List tasks of the activity] **2.** ...
Completion Criterion
[(Noun : State) or (Work product : Level of detail)]
Work products associated with the completion criterion **1.** [List the work products that are evidence of meeting the completion criterion] **2.** ...

Figure 5. Activity Card.

Note. Reproduced from Modelo para la definición unificada de la práctica como constructo teórico en ingeniería de software (doctoral dissertation), by Baron, A. (2019), Universidad Nacional de Colombia, Institutional Repository UNAL. https://repositorio.unal.edu.co/handle/unal/76404

Process for defining well-formed and well-named practices. Figure 7 illustrates the process guiding the definition of well-formed and well-named practices.

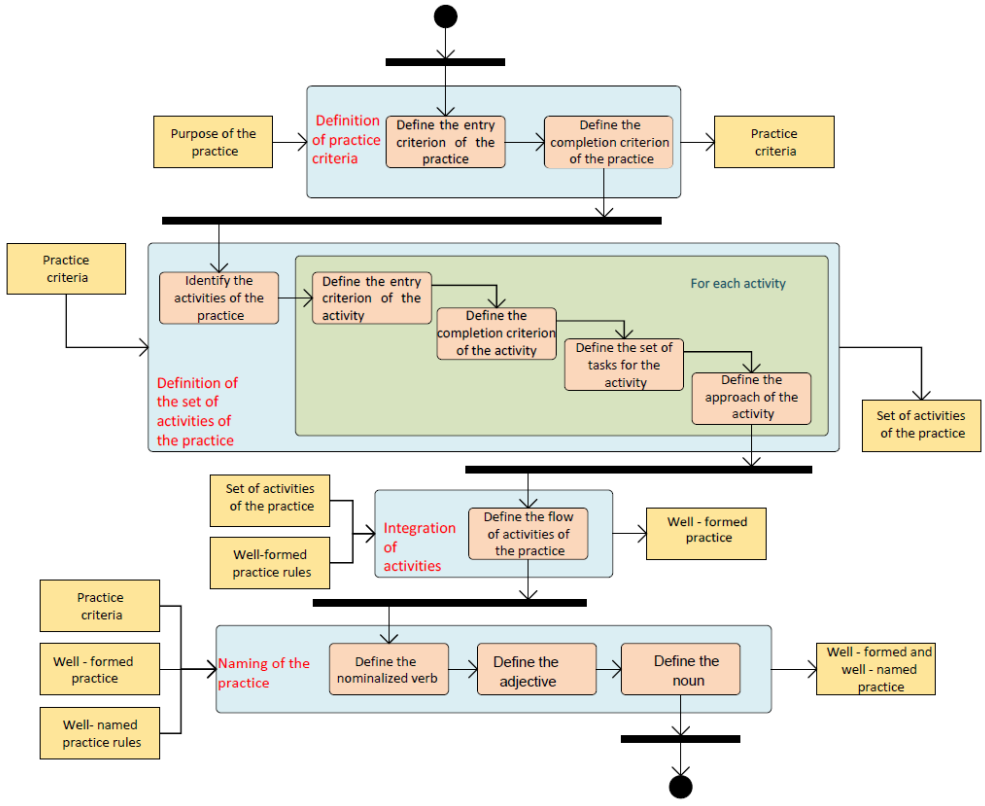

Figure 6. Process for Defining Well-Formed and Well-Named Practices.

Note. Reproduced from Modelo para la definición unificada de la práctica como constructo teórico en ingeniería de software (doctoral dissertation), by Baron, A. (2019), Universidad Nacional de Colombia, Institutional Repository UNAL. https://repositorio.unal.edu.co/handle/unal/76404

Analysis Phase – Study on the development of software assets based on reuse

¿How is the practice of development of software assets based on reuse currently defined?
Over time, the concept of reuse has been integrated into software development. Various sources address this topic from different perspectives, ranging from informal descriptions to research studies. In the SWEBOK - Guide to the Software Engineering Body of Knowledge, software reuse is described as the use of different assets, not just source code, to produce a new software solution. The process of software reuse encompasses two practices: developing assets for reuse and developing software by reusing pre-established assets (IEEE Computer Society, 2004).
Software reuse can be a broad practice that includes various concepts and areas such as domain engineering, reuse methods, and reuse metrics (Figueroa, 2010).
Solis and Hurtado (2020) reviews several reuse approaches, such as model-driven engineering, component-based development, and service-oriented architecture. According to Polo Usaola

(2013), model-driven engineering uses a conceptual model to support reuse, for example, entity–relationship schemas for databases. Additionally, Nikolaidis et al. (2024), mentions that reuse methods include both generative and compositional approaches. Generative methods focus on automating parts of the software development process and include strategies such as language-based systems, application generators, and transformational systems. In contrast, compositional methods, more common in practice, rely on reusing previously developed components in new software projects. Compositional reuse involves several stages: identifying reusable components, describing them, retrieving them, adapting them to specific needs, and integrating them into the target system (Figueroa, 2010).

Moreover, Nikolaidis et al. (2024), states that software reuse involves two key steps: identifying reusable assets, an often-complex task due to the large number of undocumented assets available and adapting these assets to the target system, a task that depends on the asset's internal structure and maintainability. Similarly, Hussein and Nouacer (2022), affirms that software reuse requires the selection of reusable components from a repository and their integration into the new development.

The selection of components for reuse may present significant challenges. In this regard, An et al. (2021), proposes a semantic model based on ontologies to enable the automatic portability of projects within the development environment, Lozano Tello and Gómez-Pérez (1998), establishes criteria for selecting reusable components, including production time, cost, final product quality, and development risk.

Additionally, Belfadel et al. (2022), Chebanyuk (2022), propose that selected assets for reuse should be aligned with and compared to the software requirements as a mechanism to ensure compatibility during integration.

Practices requiring improvement. Software engineering practices are often defined in varied and unstructured ways. It is common to find multiple approaches to applying the same practice, without clearly establishing the flow of resources between activities, a specific workflow with defined tasks, or traceability and the impact of each activity.

This lack of structure is particularly evident in the development of software assets based on reuse. Although many models provide ways to measure reuse, there are no specific guidelines to support the process or to inform decisions regarding which components to select and reuse. This practice is usually described in a textual and narrative format, which hinders its implementation in real-world contexts. Currently, there is no mechanism that outlines the necessary and sufficient elements for effective software asset reuse. As a result, reuse is often applied in an inexperienced manner, Bibi et al. (2023), based on trial and error, and without a structure that ensures proper understanding and application.

Proposed solution. This article introduces the application of the Model for Unified Definition of Practices in Software Engineering to the development of software assets based on reuse. Using this model, the essential elements are identified and defined to create a well-formed and well-named practice that is easy to understand and apply. The essentialization of the practice ensures its application is straightforward and structured, providing critical elements for developing software with reuse.

Construction Phase – Essentialization of the Development of Software Assets Based on Reuse Practice

To essentialize the development of software assets based on reuse practice, four steps established in the model for defining practices in software engineering were applied.

Definition of Practice Criteria. This step involved two activities:

Defining the entry criterion for the practice. The entry criterion for the practice is defined as a pair consisting of a noun and its state: (Software assets: reusable).

Defining the completion criterion for the practice. The completion criterion for the practice is similarly defined as a pair: (Software with reuse: integrated).

Definition of the Set of Practice Activities. This step involved five activities:

Identifying the activities of the practice. The activities that make up the practice are (Figueroa, 2010):

- Selecting relevant reusable components
- Evaluating reusable components
- Adapting reusable components
- Integrating reusable components into specific software

Defining the entry criteria for each activity. The entry criteria for each activity are presented in Table 2.

Defining the completion criteria for each activity. The completion criteria for each activity are also presented in Table 2.

Table 2. Entry criteria and completion criteria for each activity.

Activity	Entry Criterion		Completion Criterion	
	noun	**State**	**noun**	**State**
Selecting relevant reusable components	Software assets	Reusable	Software assets	Relevant
Evaluating reusable components	Software assets	Relevant	Software assets	Evaluated
Adapting reusable components	Software assets	Evaluated	Software assets	Adapted
	Software-specific requirements	Established		
Integrating reusable components	Software assets	Adapted	Software with reuse	Integrated
	Software-specific requirements	Established		

Defining the set of tasks for each activity. The tasks associated with each activity are listed in Table 3 (Figueroa, 2010).

Defining the approach of each activity. The approach of each activity is also detailed in Table 3 (Figueroa, 2010).

Table 3. Tasks and approach for each activity.

Activity	Tasks	Approach
Selecting relevant reusable components	• Establish component granularity (size) • Establish component generality • Categorize relevant components	Systematic
Evaluating reusable components	• Evaluate global functionality • Assess coherence with software requirements • Assess contributions to productivity, quality, and cost improvement	Systematic
Adapting reusable components	• Review specific software requirements • Apply component adaptation techniques (e.g., substitution, inheritance, modifications)	Systematic
Integrating reusable components	• Identify relationships between components • Describe technical relationships • Integrate components into specific software • Verify the specific software • Document generalization and adaptation of reused components	Systematic

Integration of Activities. This step included:

Defining the activity workflow of the practice. Using the rules for a well-formed practice, the order of activity execution and resource transfer between activities were defined. Table 4 presents the execution order and compliance with the rules of coherence, consistency, and sufficiency.

Table 4. Execution order and compliance with rules of coherence, consistency and sufficiency.

Entry criterion of the practice: Software assets: reusable		**Practice: Systematic Reuse of Software Assets**	**Completion criterion of the practice: Software with reuse: integrated**	
Execution Order	Entry Criterion	Activity	Completion Criterion	Contribution to Practice

1	Software assets: reusable	Selecting relevant reusable components	Software assets: relevant	20%
2	Software assets: relevant	Evaluating reusable components	Software assets: evaluated	20%
3	Software assets: evaluated Software-specific requirements: established	Adapting reusable components	Software assets: adapted	20%
4	Software assets: adapted Software-specific requirements: established	Integrating reusable components	Software with reuse: integrated	20%

Naming the Practice. This step included the following activities:

Defining the nominalized verb. Based on the practice's entry criterion, the nominalized verb is: Reuse.

Defining the adjective. Based on the consolidated set of adjectives and activity approach, the adjective is: Systematic.

Defining the noun. Based on the consolidated set of nouns and the practice's completion criterion, the noun is: Software assets.

Thus, the practice is named: Systematic Reuse of Software Assets. The practice card is presented in Figure 8, and the activity cards are shown in Figures 9, 10, 11, and 12.

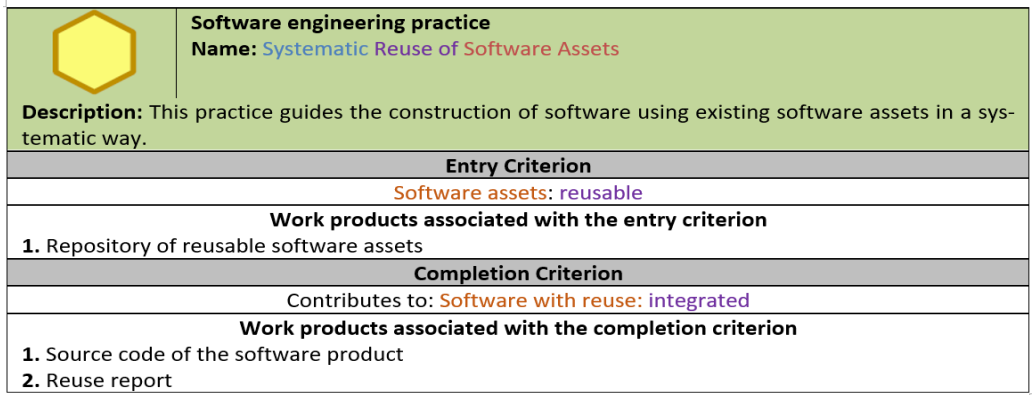

Figure 7. Practice card: Systematic Reuse of Software Assets.

Activity [1]: Selecting relevant reusable components **Approach:** Systematic **Activity Space:** Implement the system
Description: This activity allows extracting the necessary and sufficient components from the reusable asset repository that can be adjusted to the software's needs.
Entry Criterion
Software assets: reusable
Work products associated with the entry criterion
1. Repository of reusable software assets
Tasks
1. Establish the granularity of reusable components (size) **2.** Establish the generality of reusable components **3.** Categorize relevant reusable components
Completion Criterion
Software assets: relevant
Work products associated with the completion criterion
1. List of selected components

Figure 8. Activity Card: selecting relevant reusable components.

Activity [2]: Evaluating reusable components **Approach:** Systematic **Activity Space:** Implement the system
Description: This activity allows verifying whether the selected reusable components function correctly globally, whether they are consistent with the software requirements, and whether their adaptation contributes to productivity, quality, and cost improvements in software.
Entry Criterion
Software assets: relevant
Work products associated with the entry criterion
1. List of selected components
Tasks
1. Evaluate global functionality **2.** Evaluate consistency with software requirements **3.** Evaluate contribution to productivity, quality, and cost improvements in the software
Completion Criterion
Software assets: evaluated
Work products associated with the completion criterion
1. Evaluation report document

Figure 9. Activity Card: Evaluate Reusable Components.

Activity [3]: Adapting reusable components **Approach:** Systematic **Activity Space:** Implement the system
Description: This activity allows adjusting the selected and evaluated software assets to the specific needs of the software.
Entry Criterion
Software assets: evaluated specific requirements: established
Work products associated with the entry criterion
1. Evaluation report document
Tasks
1. Review specific software requirements **2.** Apply component adaptation techniques (substitution, inheritance, modifications)
Completion Criterion
Software assets: adapted
Work products associated with the completion criterion
1. Software assets adapted

Figure 10. Activity Card: Adapting reusable components.

	Activity [4]: Integrating reusable components **Approach:** Systematic **Activity Space:** Implement the system
Description: This activity allows using the adapted software assets in the construction of specific software.	
Entry Criterion	
Software assets: adapted specific requirements: established	
Work products associated with the entry criterion	
1. Software assets adapted	
Tasks	
1. Identify relationships between components **2.** Technically describe the relationships between components (diagrams) **3.** Integrate components into the specific software **4.** Verify the specific software **5.** Document the generalization and adaptation of reused components	
Completion Criterion	
Software with reuse: integrated	
Work products associated with the completion criterion	
1. Source code of the software product 2. Reuse report	

Figure 11. Activity Card: Integrating reusable components.

Validation Phase – Expert Evaluation of the Essentialized Practice

To demonstrate that the practice was correctly essentialized, it was evaluated by experts using the focus group technique. Focus group research is a strategy for obtaining qualitative data from experts on a specific topic, as a form of interview that utilizes communication between the researcher and participants, with the purpose of obtaining feedback on the topic (Hamui-Sutton & Varela-Ruiz, 2013). This type of expert-based validation has been widely used in different research contexts to assess the clarity, usefulness, and applicability of methodological proposals and technological solutions, supporting its relevance as a qualitative evaluation strategy (Enríquez Martínez & Insuasti, 2025; Dulce-Villarreal et al., 2025). The experts were carefully selected based on predetermined criteria of expertise in the subject. The steps followed in this process are illustrated in Figure 13.

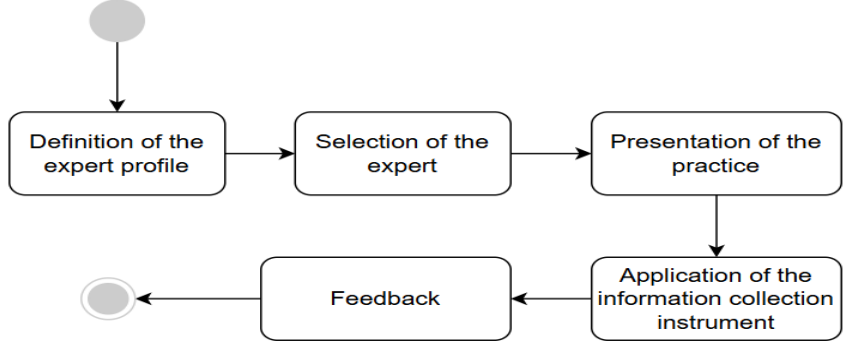

Figure 12. Activities to Validate the Essentialized Practice.

Application of the validation process. The expert selected to validate the essentialized practice was Jesús Homero Insuasti Portilla, Ph.D., an expert in Software Engineering, Software Development, and the Essence Kernel. A survey was used as the information collection mechanism, consisting of six specific questions and one open-ended question for general suggestions. The following observations were obtained:

- The expert considers that the activities contribute substantially to the practice's completion criteria, making them coherent.
- Based on Semat's consistency rule, the expert states that the activities are consistent, as the practice's articulating elements are coherent and clear relationships between activities are evident.
- The expert finds the activities of the practice sufficient, as their execution ensures the practice's completion criteria are met.
- Regarding the naming of the practice, the expert suggests changes to the verb and adjective, noting that reuse is understood as an adjective indicating the focus on development based on reuse. The expert recommends naming the practice: Development of Software Assets Based on Reuse.
- The expert considers the practice easy to understand due to its clarity and readability, provided the person applying it has technical knowledge of the subject. However, to make it understandable for non-technical individuals, the expert suggests avoiding technical terms in the activity cards.
- The expert believes the representation mechanism for the practice facilitates its application.
- The expert suggests analyzing and considering changes to the name, reviewing the concepts presented in the cards, and creating a diagram to visually represent the application of the practice.

To determine the validation outcome and establish whether the practice is well-formed and well-named, a percentage weight was assigned to the six survey questions. Validation criteria are defined as follows in Table 5:

Table 5. Execution order and compliance with rules of coherence, consistency and sufficiency.

Validation Criteria	
1% - 50%	The practice is not correctly essentialized and requires substantial changes. Validation is not approved.
50% - 80%	The practice requires some changes to approve validation.
80% - 100%	The practice is correctly essentialized, requiring no or minimal changes. Validation is approved.

According to the survey results presented in Table 6, the practice meets the third criterion: it is correctly essentialized, requiring no or minimal changes. Validation is therefore approved.

Table 6. Results of the validation of the essentialized practice.

Validation results	
Question	Percentage
1	16,6%
2	16,6%

3	16,6%
4	10%
5	15%
6	16,6%
Total	91.4%

Based on the expert's observations, the following changes were made to the essentialization of the Systematic Reuse of Software Assets practice:

- Practice name: The new name for the practice is: Development of Software Assets Based on Reuse.
- Practice focus: The new focus of the practice is: based on reuse.
- Definition of a process to guide practice application: A process was defined to graphically indicate the elements necessary to apply the Development of Software Assets Based on Reuse practice, as shown in Figure 14.

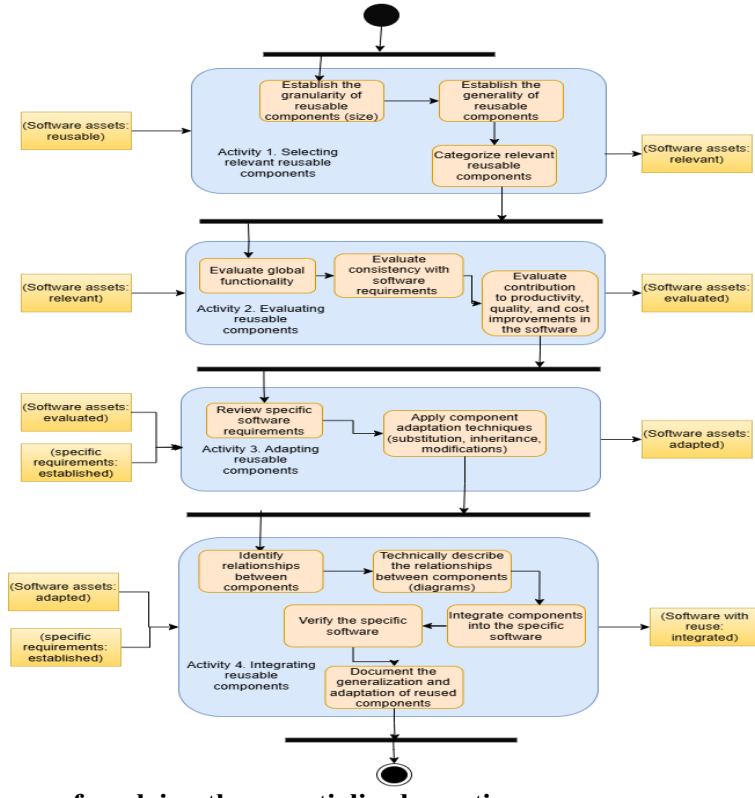

Figure 13. Process of applying the essentialized practice.

Contributions and Opportunities for Improvement

The essentialization of the practice of developing software assets based on reuse enabled the identification and structuring of the necessary and sufficient elements for its coherent application. Among the most relevant results is the clear definition of activities, tasks, work products, and input and output criteria, which facilitates the understanding, application, and evaluation of the practice in real software engineering contexts.

Unlike traditional approaches where practices are presented narratively or with little structure,

the use of the Model for the Unified Definition of Practices helped overcome the ambiguity commonly found in the documentation of reuse practices. This contribution represents a significant advancement, as it provides a framework that offers methodological clarity and enables the objective evaluation of the practice. The expert validation supports the usefulness and applicability of the proposed approach, especially in terms of the clarity and appropriateness of the defined elements.

Compared to other approaches found in the literature, such as Model-Driven Engineering (MDE), this proposal does not focus on a specific technical paradigm, but rather on the structured formalization of a practice that can be integrated into various methodological environments. While MDE uses conceptual models to facilitate reuse, for example, through model transformations or automatic code generation, essentialization focuses on the procedural and conceptual definition of a practice, regardless of the tools used. Thus, both approaches are complementary: MDE enhances automation and abstraction, whereas essentialization improves operational clarity and standardization.

Among the main advantages of the proposed approach is the possibility of replicating the practice in different contexts, thanks to its clear and modular structure. In addition, the applied model promotes traceability between activities, their objectives, and the expected outputs. One opportunity for improvement lies in further formalizing the relationships between the essential elements of the practice to facilitate integration with automated tools or adaptation to agile methodologies. Furthermore, the validation process could be enriched through larger-scale empirical studies involving development teams in real-world scenarios.

Finally, an open research area involves analyzing the impact of practice essentialization on software quality and development processes. Future studies could explore the longitudinal application of this practice in real projects and assess metrics such as efficiency, maintainability, and defect reduction, to consolidate its practical and scientific value.

Conclusions

This article presented the essentialization of the Development of Software Assets Based on Reuse practice by applying the Model for the Unified Definition of Practices in Software Engineering. The essentialization process allowed for the definition of the necessary and sufficient elements to make the practice easy to understand and apply. Additionally, it facilitated a more structured and coherent representation of the practice, contributing to its adoption in various Software Engineering contexts.

The validation conducted using the expert evaluation technique provided valuable feedback for refining the essentialization of the practice. The adjustments made ensure that the practice is well-defined, properly named, and aligned with the principles of the applied model.

Based on this experience, opportunities have been identified to further enrich the methodology used. In particular, a deeper formalization of the relationship between the essential elements of the practice and its applicability in various development environments could be explored. Additionally, strengthening the evaluation criteria would contribute to a more detailed characterization of the complexity of the essentialized practice and its impact on software quality. Furthermore, this research lays the groundwork for future studies that could expand its scope. A key next step would be to apply the essentialized practice in real-world scenarios of systematic software asset reuse to assess its impact on productivity and software quality. Additionally, it is recommended to broaden the validation process through more extensive empirical studies, incorporating feedback from professionals in various software development domains. Finally, there is an opportunity to continue evolving Barón's model by exploring enhancements that

further strengthen its applicability to emerging practices in Software Engineering.

Authors' Contributions

This article is part of the research work developed within the master's thesis entitled "Extensión del Núcleo Essence para la Construcción Colectiva del Pensamiento Estratégico de Facultad en la Universidad de Nariño." Erika Dayana Orbes Gustin was the author of the thesis, leading the conceptualization, design, and development of the research, as well as the writing of the manuscript. Dr. Alexander Barón Salazar served as the thesis advisor, contributing to the methodological guidance, conceptual structuring, and critical review of the manuscript. Dr. Oscar Revelo Sánchez participated as an academic reviewer, supporting the final review and validation of the content. All authors made substantial contributions to both the research and the manuscript, meeting the established authorship criteria.

AI Statement

During the preparation of this research, the authors used the ChatGPT language model developed by OpenAI to assist with writing, editing, and improving the manuscript's style. After using this tool, the authors carefully reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

Adjandra, W., Putrapratama, Y. B., Wiraguna, A., Sensuse, D. I., & Safitri, N. (2021). Revisión sistemática de la literatura: Reutilización del conocimiento en el desarrollo de software. *In Proceedings of the 2021 International Conference on Computer Science and Engineering (IC2SE)* (pp. 1–7). IEEE. https://doi.org/10.1109/IC2SE52832.2021.9792093

An, Y., Qin, F., Chen, B., Simon, R., & Wu, H. (2021). OntoPLC: Semantic model of PLC programs for code exchange and software reuse. *IEEE Transactions on Industrial Informatics, 17(3),* 1702–1711. https://doi.org/10.1109/TII.2020.2997360

Baron, A. (2019). *Modelo para la Definición Unificada de la Práctica como Constructo Teórico en Ingeniería de Software* [Doctoral dissertation, Universidad Nacional de Colombia]. Institutional Repository UNAL. https://repositorio.unal.edu.co/handle/unal/76404

Belfadel, A., Laval, J., Bonner Cherifi, C., & Moalla, N. (2022). Requirements engineering and enterprise architecture-based software discovery and reuse. *Innovations in Systems and Software Engineering, 18(1),* 39–60. https://doi.org/10.1007/s11334-021-00423-5

Bibi, N., Rana, T., Maqbool, A., Afzal, F., Akgül, A., & De la Sen, M. (2023). An intelligent platform for software component mining and retrieval. *Sensors, 23(1),* 525. https://doi.org/10.3390/s23010525

Chebanyuk, O. (2022). An approach to software assets reusing. In T. Zlateva & R. Goleva (Eds.), *Computer science and education in computer science* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 450, pp. 73–83). Springer. https://doi.org/10.1007/978-3-031-17292-2_6

Dulce-Villarreal, E., Hernandez, G., Insuasti, J., Hurtado, J., & Garcia-Alonso, J. (2025). Validation of MUISCA: A MDE-Based Tool for Interoperability of Healthcare Environments Using Smart Contracts in Blockchain. *Studies in health technology and informatics, 323,* 255–259. https://doi.org/10.3233/SHTI250090

Figueroa, C. (2010). *Reutilización de software, metodologías y aplicaciones más frecuentes* [Undergraduate thesis, Universidad de San Carlos de Guatemala].

Gómez Álvarez, M. C., Sánchez-Dams, R., & Barón Salazar, Á. A. (2018). A representation proposal of practices for teaching and learning software engineering using a SEMAT

kernel extension. *Revista Ingenierías Universidad de Medellín, 17(32),* 129–154. https://doi.org/10.22395/rium.v17n32a7

Guo, Z., Chen, S., Xu, X., & Chen, X. (2024). PC-TRT: A test case reuse and generation tool to achieve high path coverage for unit test. *SoftwareX, 28,* 101918. https://doi.org/10.1016/j.softx.2024.101918

Hamui-Sutton, A., & Varela-Ruiz, M. (2013). La técnica de grupos focales. *Investigación en Educación Médica, 2(5),* 55–60. https://doi.org/10.1016/S2007-5057(13)72683-8

Hussein, M., & Nouacer, R. (2022). Reuse-based agile development process for drone software systems. *International Journal of Software Engineering and Knowledge Engineering, 32(3),* 347–362. https://doi.org/10.1142/S0218194022500255

IEEE Computer Society. (2004). *SWEBOK V3.0: Guide to the software engineering body of knowledge.* https://www.computer.org/education/bodies-of-knowledge/software-engineering

Jacobson, I., Ng, P. W., McMahon, P. E., Spence, I., & Lidman, S. (2012). *The essence of software engineering: The SEMAT kernel.* Addison-Wesley.

Lozano Tello, A., & Gómez-Pérez, A. (1998). *Factores de decisión para la reutilización de componentes software* (Informe técnico). https://oa.upm.es/72622/1/CON_NAC_06.pdf

Martínez, H. D. E. ., & Insuasti, J. . (2025). Artificial intelligence and vehicle license plate recognition: A literature review. *Edelweiss Applied Science and Technology, 9(2),* 1967–1979. https://doi.org/10.55214/25768484.v9i2.4984

Nikolaidis, N., Arvanitou, E., Volioti, C., Maikantis, T., Ampatzoglou, A., Feitosa, D., Chatzigeorgiou, A., & Krief, P. (2024). Eclipse Open SmartCLIDE: An end-to-end framework for facilitating service reuse in cloud development. *Journal of Systems and Software, 207,* 111877. https://doi.org/10.1016/j.jss.2023.111877

Object Management Group. (2014). Kernel and language for software engineering methods (Essence) (Version 1.0). https://www.omg.org/spec/Essence/1.0/

Orbes, E., Barón, A. & Revelo, O. (2024, October 3-4). *Systematic Reuse of Software Assets: An Essentialized Practice in Software Engineering.* 7° Congreso Internacional de Ingenierías, Universidad Politécnica Estatal del Carchi, Tulcán, Ecuador.

Polo Usaola, M. (2013). *Desarrollo de software basado en reutilización.* Universitat Oberta de Catalunya.

Solis, A., & Hurtado, J. (2020). Reutilización de software en la robótica industrial: Un mapeo sistemático. *Revista Iberoamericana de Automática e Informática Industrial, 17(4),* 354–363. https://doi.org/10.4995/riai.2020.13335

Zapata Jaramillo, C. M., & Henao Roqueme, A. (2021). A proposal for improving the Essence standard by using terminology unification. *Ingeniería, 26(2),* 213–232. https://doi.org/10.14483/23448393.16428