electronics

Article

Agile Software Development and Reuse Approach with Scrum and Software Product Line Engineering

Wen-Tin Lee * **and Chih-Hsien Chen**

Department of Software Engineering and Management, National Kaohsiung Normal University, Kaohsiung 802, Taiwan; m10774711@mail.nknu.edu.tw
* Correspondence: wtlee@mail.nknu.edu.tw

Abstract: Agile methods and software product line engineering (SPLE) are widely recognized as practical approaches for delivering high-quality software, adapting to evolving stakeholder needs, and tackling complex problems. This study proposes a hybrid agile software development and reuse approach called SPLE-Scrum based on the activities of software product line engineering and Scrum. Within the SPLE process, we incorporate requirement engineering and design practices to create a reference architecture with reusable components called core assets by introducing a product management meeting. The core assets are reused to build a series of applications with various product lines. The product increments are delivered in each Sprint with the review and retrospective meetings based on Scrum lifecycle and practices. We present a case study involving a blockchain online store to demonstrate the practical application of SPLE-Scrum, highlighting the benefits of integrating Scrum and software product line engineering. The research hypotheses of the proposed approach were validated through a study of structured interviews with 5 experts and 44 software practitioners, showing that the key factors of product management, project requirements, and product architecture in the SPLE-Scrum approach have a beneficial impact on project success. The SPLE-Scrum approach provides valuable insights and practical guidance for organizations seeking to optimize their software engineering practices while incorporating agile development and software reuse capabilities.

Keywords: agile methods; Scrum; software product line engineering; software reuse; software development

check for updates

Citation: Lee, W.-T.; Chen, C.-H. Agile Software Development and Reuse Approach with Scrum and Software Product Line Engineering. *Electronics* **2023**, *12*, 3291. https://doi.org/10.3390/electronics12153291

Academic Editor: Mehdi Sookhak

Received: 26 June 2023
Revised: 20 July 2023
Accepted: 26 July 2023
Published: 31 July 2023

1. Introduction

Software development methods aim to increase the development team's productivity, shorten the time to market, reduce development costs, and improve customer satisfaction. To achieve the above goals, agile software development, also known as agile development, has gradually aroused public discussion since the 1990s. It advocates adaptive planning and evolutionary development, shares the same software process values, and encourages rapid and flexible responses to changes through early delivery and continuous improvement [1]. Agile development emphasizes moderate planning, human-oriented cooperation, face-to-face communication, self-organization and management, and rapid development [2]. Software organizations adopt large-scale agile practices [3–6] to replicate the success of agile methods on team projects at the organizational level.

Although the agile software development method can tolerate changes in requirements and can effectively and quickly solve problems, increase output, and shorten the entire development timeline, it also has some shortcomings which may lead to project failures. For example, requirement identification and initial planning is the first challenge of Scrum [7]. Another challenge of Scrum is the lack of attention to design. Scrum pays less attention to requirement engineering in the analysis and design phase. In addition, its lack of traceability of documents and files and configuration management may affect product quality or lead to project failure. Software product line engineering (SPLE) [8–11] uses product-line methods to produce products for customers with different needs. Software product-line engineering aims to develop software products at a minimal cost [8,12] and improve software quality by increasing the reuse of existing software components [13].

A hybrid agile approach [14] combines agile methods and non-agile technologies, while a blended agile approach combines two or more agile methods. This study aims to address the issues present in the agile software development methods mentioned above through a hybrid method approach to achieve the following two objectives: 1. To compare and explore the differences between the SPLE and Scrum methods from engineering, quality assurance, and project management perspectives. 2. To address and improve potential issues inherent in Scrum, such as enhancing requirement engineering during the analysis and design phases, establishing traceability of artifacts, implementing configuration management, etc. Therefore, this work combines Scrum with SPLE to propose an agile development and reuse approach named SPLE-Scrum to improve requirement engineering, software design and reuse, and software artifacts' traceability by integrating these methods' distinct and viable technological advantages.

Based on the SPLE-Scrum approach, the key factors of "product management", "project requirements", and "product architecture" [15] are considered independent variables to measure the degree of agreement among questionnaire respondents regarding the potential realization of project success, which is used as the dependent variable. This study proposes the following research hypotheses as the foundation for subsequent questionnaire surveys, data analysis, and verification:

- Hypothesis 1 (H1): Product management (PM) key factors of the SPLE-Scrum approach significantly impact project success (PS).
- Hypothesis 2 (H2): Project requirement (PR) key factors of the SPLE-Scrum approach significantly impact project success.
- Hypothesis 3 (H3): Product architecture (PA) key factors of the SPLE-Scrum approach significantly impact project success.

Regarding the SPLE-Scrum approach, we collected expert opinions and suggestions to validate the research hypotheses proposed in this study and whether the SPLE-Scrum method can meet or achieve the goals of key factors such as product management, project requirements, and project architecture. We used the Likert five-point scale for scoring and measurement. Before conducting expert interviews and completing the first-stage questionnaire, we provided five domain experts with an overview of this study's main motivations and objectives, the relevant problems that Scrum may have, and the design principles of SPLE-Scrum. We also presented the detailed case study of this study to provide them with a comprehensive understanding and to gather their comments and recommendations. We conducted a reliability analysis for the entire questionnaire and the result of 0.861 fell within the highly reliable range.

Through a survey questionnaire and open discussion with 44 software practitioners to validate the hypotheses of the SPLE-Scrum approach, we present our findings on how our approach benefits product management, project requirements, product architecture, and project success. The research hypotheses of the SPLE-Scrum approach are validated, showing that the key factors of product management, project requirements, and product architecture in the SPLE-Scrum approach have a beneficial impact on project success.

The sequel will outline background knowledge and related work in the next section. In Section 3, SPLE-Scrum will be explained in detail. Section 4 provides a case study using SPLE-Scrum. Finally, we summarize the potential benefits of the proposed approach and outline our future research plan in Section 5.

2. Background Knowledge and Related Work

This section introduces basic concepts of the Agile Method, Scrum, and Software Product Line Engineering (SPLE) with a brief overview of feature models and product configuration.

2.1. Agile Method and Scrum

Agile means fast, light, and dynamic. Agile methods are more effective and faster when responding to changes than traditional software development methods [16–18]. They emphasize individual interactions between processes and tools, self-organizing teams, continuous release of new software features, and customer collaboration [19]. Agile methods maintain rigorous engineering processes and adopt best practices while helping stakeholders work with software developers to build, deploy, and maintain complex software [20]. They emphasize adapting to changes rather than predicting [21]. The focal aspects of agile methods are simplicity and speed [17]. Agile software development methods generally have the following characteristics: incremental: small software releases with rapid cycles; cooperative: customers, developers, and relevant stakeholders work and communicate together closely; straightforward: the method itself is easy to adopt and well documented; and adaptive: the ability to deal with changes.

According to the Agile Status Survey [22], Scrum was reported as the most widely practiced agile methodology. At least 72% of respondents currently practice the Scrum method or a hybrid approach containing Scrum. Scrum was first introduced by Takeuchi and Nonaka [23] in the context of product development. The term Scrum is borrowed from the rugby game, which means that only by maintaining an overall forward method, like passing the rugby ball within the team, can it cope with the challenges of the current complex market [24]. The Scrum framework proposed by Jeff Sutherland and Ken Schwaber [25] is an agile method that can deal with changes by developing and reviewing software increments iteratively. Unlike the waterfall model, which breaks down project activities into different phases, Scrum focuses on developing a set of high-value features incrementally and iteratively through each Sprint to obtain customer feedback faster [26].

Figure 1 shows the Scrum process with required meetings and artifacts. The Scrum team is a small cross-functional, self-organizing team that uses iterative and incremental processes for the project or product development. Team members are responsible for creating and adapting the overall process within this structure. The management representative of the team is the Scrum Master. The primary responsibility of the Scrum Master is to eliminate obstacles to the team and ensure that Scrum practices are followed. Product Backlog is the priority list of all requirements or user stories to be implemented in the project; the Product Owner has the right to determine the priority of the user story [21]. Grooming is managing product backlog with prioritized requirements and estimating the amount of work to complete the requirements [27].

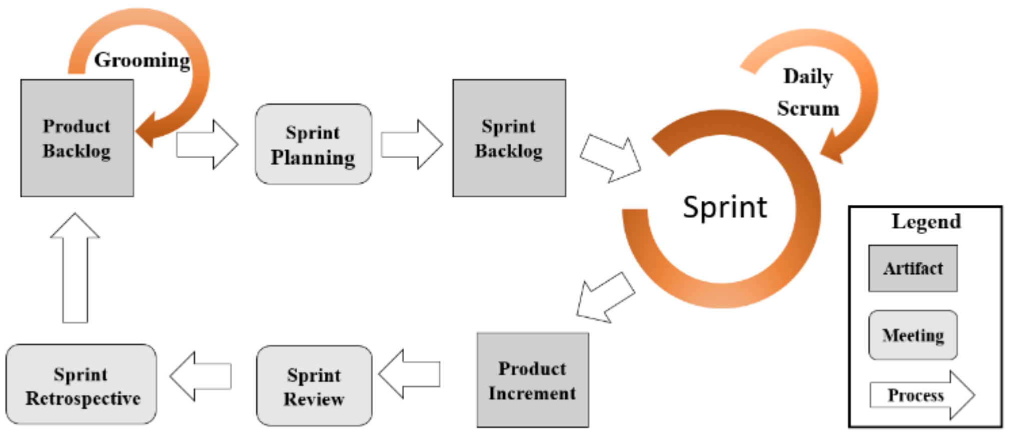


Figure 1. Scrum framework [27].

Sprint is a time-boxed development period based on product goal and complexity, usually 1–4 weeks. The Scrum team builds and tests product increments with new features that can be released in each Sprint. Sprint begins with a Sprint planning meeting where the product owners, Scrum Master, and Scrum team members determine what needs to develop. The Scrum team then uses Sprint goals in internal meetings to obtain a list of requirements in the Sprint backlog. A successful Sprint depends on whether Sprint goals and the requirements in the Sprint backlog are achieved and satisfied.

During the Sprint, the Scrum master holds a 15 min "Daily Scrum" or "Daily Standup Meeting" with the Scrum team to review project progress. Each team member will answer three questions [25]: 1. What has been done since the last meeting? 2. What will be done before the next meeting? 3. What are the obstacles in the process?

Each Sprint provides an incremental version of a potentially deliverable product. The team produces software that is coded, tested, and usable at the end of each Sprint. The Scrum team will hold a Sprint review meeting to show their results during the Sprint. Next, the Scrum team evaluates its work and processes in a Sprint retrospective meeting to prioritize improving the team's processes before the next Sprint [28].

Agile software development methods also have some challenges. For example, the challenges of Scrum are how to identify requirements, how to conduct preliminary planning, and the lack of focus on design [29]. Scrum pays less attention to requirement engineering in the analysis and design phase [30]. In addition, it has poor traceability in document archives, incomplete version control, and configuration management, all of which may be potential factors that lead to project failure and even affect subsequent system maintenance and requirements changes.

2.2. Software Product Line Engineering

Software reuse involves creating new software from existing software products that improve product quality by combining reliable and quality software components. A software product line (SPL) is a set of software-intensive systems that share features or functions generated from a group of pre-defined and reusable shared core assets [10,31]. Software product line engineering (SPLE) [8–11] uses product line methods to produce products for customers with different needs. A product line is created by combining commonalities to efficiently produce products by integrating or reusing shared core assets to meet customer needs. Large-scale customization is transparent in software product line engineering, customers can obtain unique products through their specific needs, and their common needs will be evaluated before production starts [31].

Software product line engineering aims to develop software products by reusing existing software components [13]. Different techniques and methods [32–37] can be used to develop various software products in multiple domains. Several studies [38–42] explore how to integrate agile methods and software product line engineering. There are two complementary development processes in software product line engineering: the domain engineering process and application engineering process. The domain engineering process defines and realizes the commonality and variability characteristics of the product line. Its purpose is to develop the shared core software assets and a common and reusable product line platform to promote the systematic and consistent reuse of all finished products and components. The application engineering process binds the product line's variability according to specific applications' needs. It builds a single application product or a series of product applications by reusing shared core software assets, products, or product components from domain engineering [43]. The domain engineering process comprises five sub-processes: product management, domain requirement engineering, domain design, domain realization, and domain quality assurance [8,38]. Developers identify domain variability models and implement and test reusable domain artifacts in a product line platform. The application engineering process comprises four sub-processes: application requirement engineering, application design, application realization, and application testing. Developers design application variability models and build application artifacts according to customers' needs.

3. SPLE-Scrum: An Agile Software Development and Reuse Approach

This study proposes a hybrid agile software development and reuse approach called SPLE-Scrum based on the activities and work products of SPLE and Scrum. Figure 2 shows the SPLE-Scrum process, which contains the steps Product Management meeting, Domain and Application Requirement Engineering of the Pre-Sprint, Sprint Planning, Domain and Application Design in the Sprint, Product Increment, Sprint Review, and Sprint Retrospective. The product management meeting establishes market strategy goals, product backlogs, and product roadmap with a Product Backlog Grooming mechanism. We used requirement engineering and design processes in the domain and application engineering of SPLE to create a reference architecture with reusable components called core assets. The core assets establish the commonality and variability of the products. Domain engineering assets are reused to build a series of applications with various product lines.

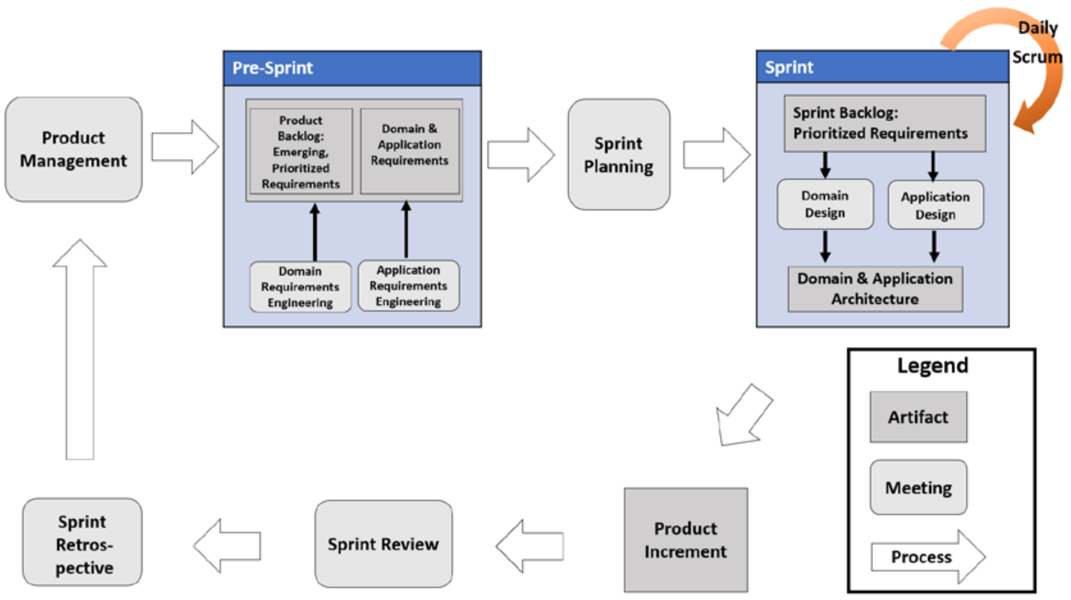

Product Management
Pre-Sprint
Product Backlog: Emerging, Prioritized Requirements
Domain & Application Requirements
Domain Requirements Engineering
Application Requirements Engineering
Sprint Planning
Sprint
Daily Scrum
Sprint Backlog: Prioritized Requirements
Domain Design
Application Design
Domain & Application Architecture
Legend
Artifact
Meeting
Process
Product Increment
Sprint Review
Sprint Retros-pective

Figure 2. SPLE-Scrum: an agile software development and reuse approach.

The proposed approach is demonstrated through a case study of a blockchain online store developed based on the steps in Figure 2. The meetings and artifacts in each step are detailed below.

3.1. Product Management Meeting

The product management meeting is scheduled before the Pre-Sprint. In the Scrum framework, the initial clarity of the product scope may be limited, and cost control can pose challenges [44]. An economic perspective from the company or enterprise level should be adopted, wherein the product production scope is defined, and market strategy objectives are emphasized to produce diversified and differentiated products while maintaining cost-effectiveness. The inputs of the product management meeting consist of the goals and milestones defined by the management team, customer, and relevant stakeholders. The outputs typically include a product roadmap, which provides a reusable list of products or development tools that can be implemented within established platforms. The Scrum team can utilize the methods of the customer journey map, experience map [45], and user story mapping [46] to elicit user needs, clarify product scope, and develop the product roadmap.

The Product Owner and the Development Team discuss the product backlog items [25], provide more details, estimate effort or complexity, and identify dependencies or potential risks. The purpose is to refine the items to a well-understood level that can be readily developed for the following phases. After completing a Sprint retrospective meeting of a Sprint, the Product Owner and the Development Team collaborate to refine the Product

Backlog. This involves reviewing, clarifying, and prioritizing product backlog items to ensure they are ready for inclusion in future Sprints.

Variability of Domain Requirements

The variability part of the MFR is first analyzed by dividing the variability part of the domain requirements into two levels [12]. The first level determines which part of the domain requirements is variability, which MFRs are common, and which are optional. For example, if we decide that most systems should provide the function [customer transaction payment] in MFRs, then it should be a common MFR. As for this common MFR, it is possible to find other different variation points, and then we can identify them more clearly from the second level.

The second level looks for the variability part and implements it as a detailed variation point. We find the variation points from two aspects: One is the constituent elements of MFR itself, divided into dynamic elements from the perspective of time and static elements from the perspective of structure. The other is from the variability types to implement it as a more detailed variation point. The variation points identified in the domain requirements can be grouped into four types: data, control, computation, and external computation. The cardinality indicates how many variants will be selected for the variation point.

Table 1 shows the difference and combination of variation points of MFR. Tax calculation variation point is common in general systems, but the implementations of each system are different because of the tax system used. Regarding transaction payment variation points, some systems will support types (2), (4), and (5), and others will support types (1), (3), or (1), (5), which will lead to differences and combinations between different types.

Table 1. Variation points of MFR.

Variation Point	**Tax Calculation**	**Transaction Payment**
Type	Operation	Control
Cardinality	[1]	[1..n]
Variant	(1) National Tax (2) Local taxes	① Bitcoin ② IOTA ③ Cardano ④ OmiseGO ⑤ Ethereum

3.2. Domain and Application Requirements Engineering of the Pre-Sprint

In the Pre-Sprint, we adopt the same domain requirement engineering approach as SPLE and also deal with the application engineering requirements of specific products [8–11]. In SPLE, the domain and application engineering process sequence has no particular direction or restriction [43]. The incremental method develops common domain products and then develops variability products based on specific customer needs in application engineering. The domain and application requirements engineering sub-processes can improve the traceability management of Scrum in requirement analysis, requirement changes, and related documents.

The Product Owner, Scrum Master, and Development Team will work together to produce the domain and application variability models covering the entire system and application scenarios. At the beginning of the domain requirement analysis, the domain requirements variability models are used to illustrate which requirement attributes or elements are public or selected. The model can further explain the type of variation point, variant, and cardinality, which will help developers to identify requirement variation points.

In the domain requirement analysis, we adopted user stories [47–49] to briefly explain the roles, descriptions, and goals of using the system or service. Then, we constructed the main functional requirements accordingly. The main functional requirement (MFR) is central to identifying and specifying domain requirements. It is the basic unit of functions in our domain requirements. Based on the variability model of domain requirements, the development of domain and application requirements of the blockchain online store is divided into the following steps, which will serve as the core assets in the Pre-Sprint.

3.2.1. Identify Domain Requirements

The Scrum team gathers the user's needs using the requirement elicitation methods such as the customer journey map and experience map [45]. The team then describes the user's needs in the form of a user story after agreeing on the scope of the domain requirements in the product management meeting. The user story with acceptance criteria [47] is recommended by Behutiye et al. [50] to document quality requirements [46]. Table 2 shows the excerpt of the user story descriptions of the blockchain online store. The enroll epic has three user stories that have their acceptance criteria. The definition of done is applied to all user stories after completing each user story.

Table 2. User story descriptions of blockchain online store (excerpt).

User Type	**User** **Epic** **Enroll**
User Story ID 2	As a customer, I want users to register as members when using this system, and if they are not, to register and then add registration data to the database so that I know who my members are.
Acceptance Criteria	1. The user can use the sign-up page, enter a username and password, and click on sign-up to complete registration. 2. System generate success and failure message after processing
User Story ID 3	As a customer, I want users to make profile changes after logging in so that membership data can be kept up to date.
Acceptance Criteria	1. The user can use the profile page and enter his profile data and click on save to complete editing. 2. System generate success and failure message after processing
User Story ID 4	As a customer, I hope that if the user forgets the password by entering the account number, and mailbox, the system will send the password to the registered mailbox so that members will not repeat the application for a new account.
Acceptance Criteria	1. The user can use the password-finding page to enter the account number and mailbox and click submit to complete the request. 2. System generate success and failure message after processing
Definition of Done	1. Passing testing per acceptance criteria items 2. Passing regression testing 3. Approved by UI team 4. Able to show features in company demo

Next, we identify the domain requirements for the blockchain online store through a series of functional requirement context matrices for building and optimizing MFRs. Table 3 shows the MFR context matrix of a blockchain online store. All identified main functional requirements are listed in the left column of the matrix, and the similar legacy systems A, B, C, D, and E are arranged in the right column. In Table 3, "O" means that the MFR can be found in the existing systems A, B, C, D, or E. "X" indicates that the MFR does not exist in the system. The "Property/Ratio" column is the ratio of the number of systems with the MFR to the total number of systems. "C" represents a commonality ratio, and "P" represents an optionality ratio. For example, the five systems use MFRs login, logout, registration, modifying personal information, and customer transaction notification. Therefore, their commonality ratio is 100%. In contrast, only two out of five existing systems require MFR8 register items, so its commonality ratio is 40%. Similarly, the commonality ratio of MFR12 "Give Opinion" is 20%.

Table 3. The MFR context matrix.

MFR		Property/Ratio	A	B	C	D	E
MFR1 login		C/100%	O	O	O	O	O
MFR2 registration		C/100%	O	O	O	O	O
MFR3 modify personal information		C/100%	O	O	O	O	O
MFR4 obtain passwords		C/100%	O	O	O	O	O
MFR5 log out		C/100%	O	O	O	O	O
MFR6 add items to shopping carts		C/100%	O	O	O	O	O
MFR7 purchases products		C/100%	O	O	O	O	O
MFR8 register items		P/40%	X	O	X	O	X
MFR9 search the product		C/100%	O	O	O	O	O
MFR10 customer transaction notification		C/100%	O	O	O	O	O
MFR10-1	Line		O	O	O	O	O
MFR10-2	E-mail		X	O	O	O	X
MFR10-3	SMS		O	X	X	X	O
MFR11 view purchase logs		C/100%	O	O	O	O	O
MFR12 give opinion		P/20%	O	X	X	X	X
MFR13 customer transaction payment (Payment Service).		C/100%	O	O	O	O	O
[V1]	IOTA		X	O	O	O	O
[V2]	OmiseGO		X	X	X	X	X
[V3]	Ethereum		O	X	X	X	X

3.2.2. Establish Domain Feature Model and Variability Model

This study employs a domain feature model to establish the interrelationships between functional features in the blockchain online store. This model offers a comprehensive view of mandatory, optional, and alternative relationships among the features, enabling teams to collaborate more effectively. Using the MFR-context matrix of Table 3, a configurable blockchain online store system is specified and built, as depicted in Figure 3.

The blockchain online store must possess mandatory features such as registration, user authentication, product search, security policies, and customer transaction notification. Moreover, the system must implement a high or standard security policy (alternative relationship) and offer different blockchain transaction payment methods (alternative relationships), including IOTA, OmiseGO, and Ethereum. Figure 3 illustrates that the system adopts blockchain technology and implements a high-security policy.

From Table 3, we infer that the low commonality ratio of MFR8 register items and MFR12 give opinion are optional functional requirements. The child features of the Search the Products in Figure 3 are presented in an optional relationship. For a customer transaction notification functionality, multiple relationships must be selected; either Line, E-mail, SMS, or a different combination.

Next, this study identifies the parts that belong to the variation points in the model and uses the domain variability model to describe the parts of the variation more clearly. The domain variability model elucidates the dependencies among the variables through the dependency link. Figure 4 depicts a triangle as a variation point named "payment service" and relationships with three variations, namely IOTA, OmiseGO, and Ethereum, represented by rectangles. The dashed line between the variation point and the variation represents an optional correlation. In contrast, solid lines denote mandatory dependencies, implying that the corresponding variables must be selected. The model is also indicated by an arc marked with the reference amount of [1..1], and only one of the three variables must be selected.

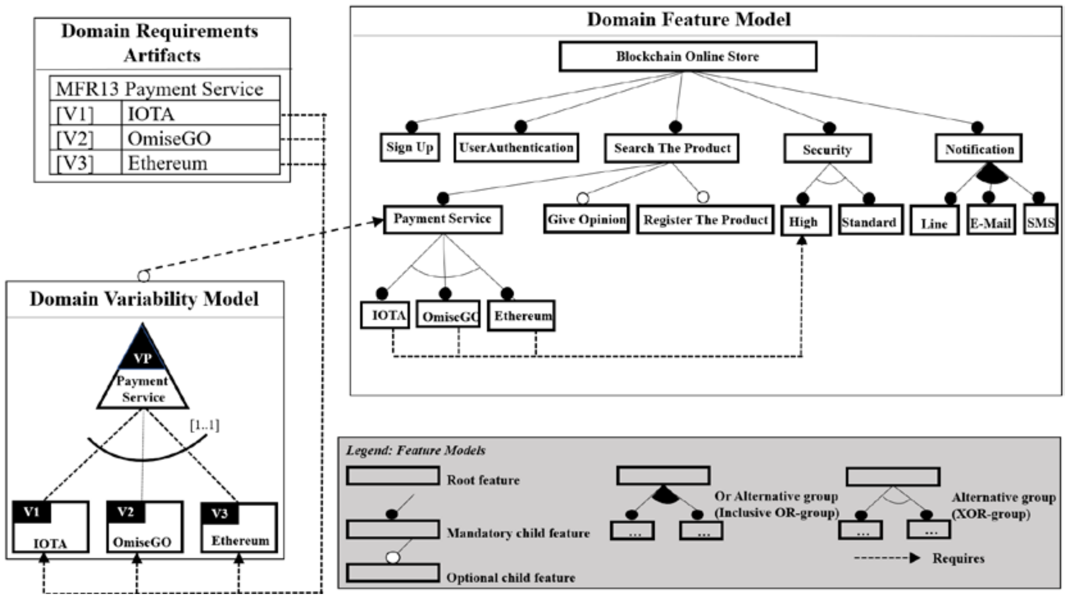


Figure 3. Domain feature model and variability model of blockchain online store.

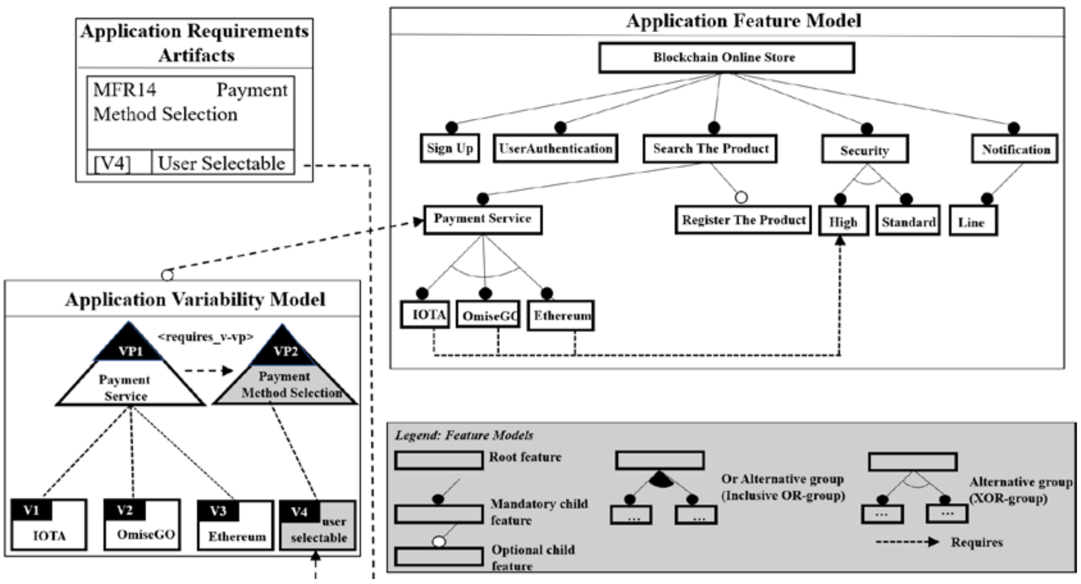


Figure 4. Application feature model and variability model of blockchain online store.

3.2.3. Establish Application Feature Model and Variability Model

During application requirement engineering, it is necessary to adjust the domain feature model and variability model to establish an application feature model and variability model. The customer wants to build a specific application that removes MFR12 give opinion and uses Line only in MFR 10 Notification. In addition, the customer thinks that the user should be able to freely select their preferred transaction payment method when making a customer transaction payment.

According to the specific application requirements, Figure 4 shows the Application Feature Model and Variability Model of Blockchain Online Store, which removes the sub-functional feature Give Opinion in the MFR9 search the product, and the sub-functional feature E-Mail and SMS in the MFR 10 Notification in Figure 3.

Figure 4 also adds a new variation point, Payment Method Selection, to accommodate the new requirements of Payment Method Selection in Application Requirements Artifacts. The variation point Payment Service (VP1) requires Payment Method Selection (VP2) to select a payment method. Given that this new variation point depends on the original customer transaction payment method, we must establish a relationship with it in the application engineering variability model and assign the variable V4 as user-selectable.

3.3. Sprint Planning

Sprint Planning is a collaborative meeting that occurs at the beginning of each Sprint and involves the Product Owner, Scrum Master, and the Development Team. Sprint Planning aims to determine what work will be tackled in the upcoming Sprint and how it will be accomplished. The Scrum team discusses and decides on the Sprint Goal, which represents the overall objective or purpose to be achieved by the end of the Sprint. The Scrum team develops the Sprint product backlog based on the Sprint goals in the upcoming Sprint.

In SPLE-Scrum, each Sprint planning meeting starts to review the product along with the product roadmap's current progress, goals, and completion status. The Product Owner presents the highest-priority items from the Product Backlog. The Development Team analyzes these items, asks questions, and estimates the effort required to complete them. The activities typically include splitting or decomposing large user stories into smaller, actionable tasks, refining acceptance criteria, updating estimates, and reordering the backlog based on evolving priorities or new insights. The Development Team will develop a Sprint product backlog, including the MFRs that have been prioritized and estimated. By the end of the Sprint Planning meeting, the Scrum team has created a Sprint Backlog, which includes the selected user stories, their corresponding tasks, estimates, and assignments to specific team members, and serves as a guide for the team's work throughout the Sprint to achieve the defined Sprint Goal.

Sprint Backlog of Blockchain Online Store

The Sprint backlog of the blockchain online store is presented in Table 4. The MFR column lists the MFR names of the product backlog items that will be developed in this project. The Prioritization column means prioritization levels (high, medium, low) for implementing the MFR. The Estimation column represents the effort required to implement the product backlog. The Sprint column indicates the specific Sprint in which the items will be implemented.

The Development Team collectively evaluates the MFR and user stories considering factors such as effort, complexity, technical dependencies, and associated risks. In SPLE-Scrum, techniques like story points or time-based estimates are used to estimate the effort required to complete the MFR and user stories [50]. One common scale utilized in Agile development is the modified Fibonacci sequence, which starts with 1 (e.g., 1, 2, 3, 5, 8, 13, 21, etc.) and allows for a slightly different distribution of effort levels. If the effort estimated for the MFR or user story is larger than 34 story points/person days, it is considered large and needs to be broken down. Numeric values assigned in estimation are not as significant as their relative differences. These assigned values aid in prioritization, planning, breaking down work into manageable tasks, and assessing the workload within each Sprint. The estimation assists project planning and shared understanding rather than aiming for precise time-based estimations.

The Scrum team collectively decides on including MFR and user stories in the Sprint based on capacity and backlog item priorities. Once the MFR and user stories are determined, the Development Team breaks them into smaller, actionable tasks during the Sprint planning meeting. For instance, tasks for the MFR "registration" may involve designing the account registration form, implementing server-side validation, and creating the database schema for user accounts.

Table 4. Sprint backlog of blockchain online store.

MFR		Prioritization	Estimation	Sprint
MFR1 login		H	5	1
MFR2 registration		H	3	1
MFR3 modify personal information		H	2	1
MFR4 obtain passwords		H	3	1
MFR5 log out		H	2	1
MFR6 add items to shopping carts		H	13	1
MFR7 purchase products		H	13	2
MFR8 register the products		H	5	1
MFR9 search the product		H	5	1
MFR10 customer transaction notification		M	8	2
MFR10-1	Line	M	8	2
MFR11 view purchase logs		M	5	2
MFR12 give opinion		L	5	2
MFR13 payment service		M	21	3
[V1]	IOTA	M	13	3
[V2]	OmiseGO	M	13	3
[V3]	Ethereum	M	13	3

3.4. Domain and Application Engineering Design in the Sprint

The Development Team focuses on developing and delivering increments of potentially shippable product functionality in Sprint. They work on the backlog items selected for the Sprint, aiming to complete the planned work within the time frame. The Scrum team participates in the Daily Scrum, a time-boxed meeting held daily to synchronize and align the team's work. Each team member shares progress, discusses any obstacles or challenges, and collaborates to ensure everyone is on track to achieve the Sprint Goal.

In SPLE-Scrum, the functional feature model and the orthogonal variability model [43,51] are used to establish a domain and application engineering variability model based on the domain requirements determined in the Pre-Sprint. The functional feature model can describe the design of the reference architecture and define the overall problem. The orthogonal variability model can be used to record the variability of product lines and describe the variability of the domain and application requirements engineering.

After the specific application requirements are determined, we adjust and design by selecting and combining the variability parts and building an application engineering variability model derived from the reference architecture defined by the domain design. The reference architecture provides a high-level architecture that includes the description and interface of commonality, variability, and reusable components. It is the most critical core asset to reuse components successfully and is represented with a component model.

The domain and application design artifacts provide a high-level architecture that includes the description and interface of commonality, variability, and reusable components. We construct domain and application use-case models to represent the domain and application engineering requirements. The core asset of reusing elements from the application development perspective is represented with a component model.

3.4.1. Domain and Application Use Case Models

The use-case model consists of actors, use cases, and their relationships within the domain and application engineering. The actors represent users outside the domain and application engineering boundary and can also be a system or device. The domain and application engineering use cases describe functional requirement units. Based on the domain feature and variability model from the Pre-Sprint, we construct a domain use-case model by adding <<include>> and <<extend>> relationships and labeling them as <<common>> or <<optional>>.

A domain use case may include optional functional requirements specific to that domain use case and not be shared with other domain use cases. Therefore, to explicitly indicate the optional parts within the domain use cases, we create separate domain use cases for the optional functional requirement units in the domain use case and add <<extend>> relationships to the corresponding domain use case.

Figure 5 shows the domain use-case model representing the functional features of the blockchain online store. The domain use cases Register the Product and Give Opinion extend to the domain use case Search the Product and establish their relations through the <<extend>> relationship. Furthermore, we add an <<include>> dependency relationship between the domain use cases Search The Product and Payment Service.

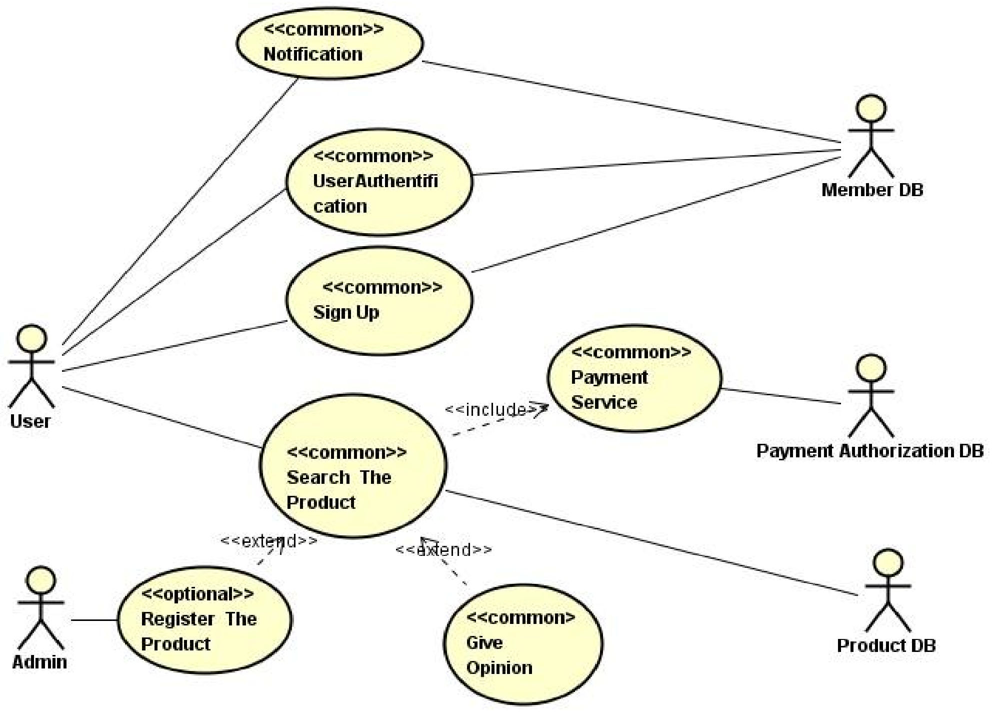
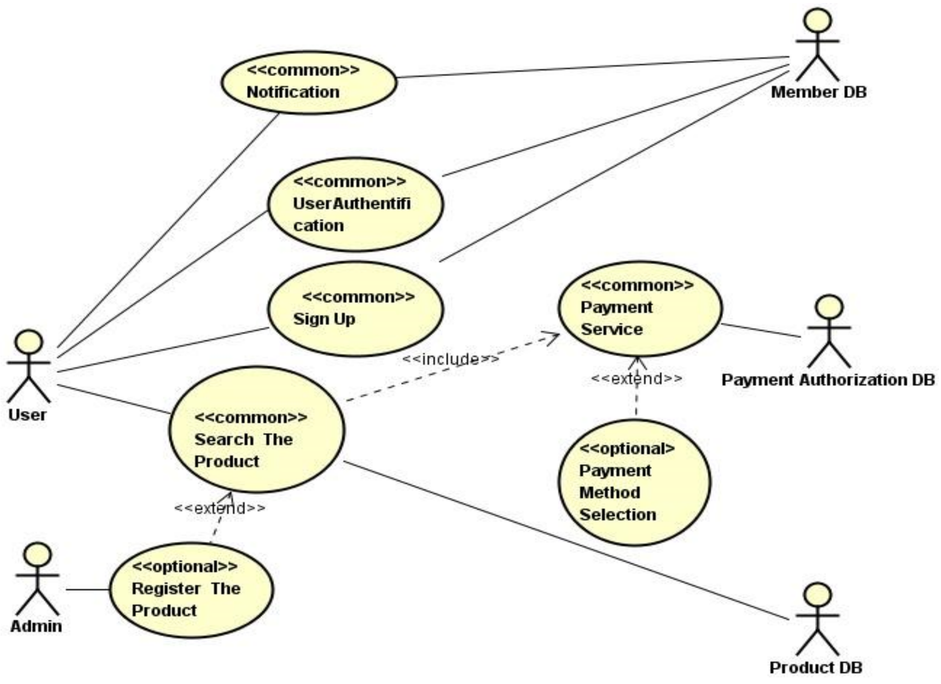


Figure 5. Domain use-case model of blockchain online store.

We adjust the design based on the specific application requirements by selecting and combining variability components derived from the reference architecture defined in the domain use-case model. The selection and combination of variability components can be accomplished by leveraging the feature and variability model generated in the Pre-Sprint, modifying the initial domain use-case model, and ultimately producing the application use-case model, which represents the final reference architecture of the entire application system. Figure 6 shows the application use-case model of the blockchain online store, which removes the domain use case Give Opinion and adds the use case Payment Method Selection.

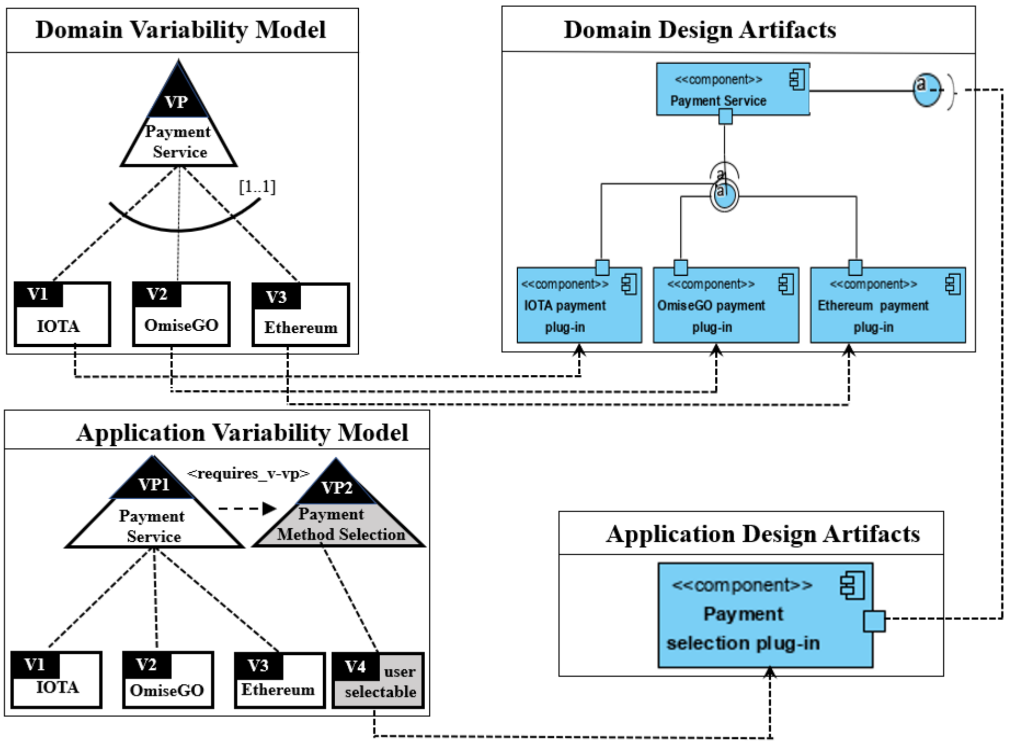


<<common>> Notification
<<common>> UserAuthentification
<<common>> Sign Up
<<common>> Payment Service
<<common>> Search The Product
<<optional> Payment Method Selection
<<optional>> Register The Product
<<include>>
<<extend>>
<<extend>>
Member DB
Payment Authorization DB
Product DB
User
Admin

Figure 6. Application use-case model of blockchain online store.

3.4.2. Domain and Application Component Model

A set of components for the blockchain online store has been developed through Sprint's domain and application design activities, which can be integrated through interfaces. Figure 7 partially extracts the blockchain online store's domain and application design artifacts, including an association with other finished products produced during the design phase.



Domain Variability Model
VP
Payment Service
[1..1]
V1 IOTA
V2 OmiseGO
V3 Ethereum
Domain Design Artifacts
<<component>> Payment Service
<<component>> IOTA payment plug-in
<<component>> OmiseGO payment plug-in
<<component>> Ethereum payment plug-in
Application Variability Model
VP1 Payment Service
<requires_v-vp>
VP2 Payment Method Selection
V1 IOTA
V2 OmiseGO
V3 Ethereum
V4 user selectable
Application Design Artifacts
<<component>> Payment selection plug-in

Figure 7. Domain and application component model of blockchain online store.

The domain variability model for customer transaction payment methods reveals three variation points related to these methods, which will be supported by the Scrum team in Sprint, such as IOTA, OmiseGo, Ethereum, etc.

An extracted component diagram represents the plug-in application of blockchain technology. When the project determines that users can freely choose payment methods during the Pre-Sprint, we delete the alternative relationship of the variants V1, V2, and V3 in the domain variability model and adjust it to a selective relationship to bind these variables to specific applications. Then, we create a payment selection plug-in component that allows for free payment method selection to the application design artifacts and link them together through the interface provided by the components of domain design artifacts to form a reference architecture in domain design, which promotes the reuse of components.

3.5. Product Increment, Sprint Review, and Sprint Retrospective

At the end of the Sprint, the increment is a completed entity that can be reviewed. The Scrum team provides incremental product functions for each Sprint. The realization of each increment is a further step towards the vision or strategic goal. In addition, regardless of whether the product owner decides to release its version, the increment must be available at any time.

The development team presents the results achieved during the Sprint review meeting, while stakeholders provide feedback at the end of a Sprint. The defects may be found and discussed during the function demo of the Sprint review. To incorporate the advantages of engineering methodologies such as extreme programming [52,53], SPLE-Scrum recommends that refactoring techniques be proposed during the Sprint review process to enhance code readability, simplify its structure, and facilitate maintenance and scalability of the implemented functions. After the Sprint review, the Sprint retrospective meeting is held to review and adjust the process to improve the team's work. The primary purpose is to let the Scrum team review the process of this Sprint, examine the roles, relationships, tools, and operations in this Sprint cycle, and think about whether the next Sprint can make more progress.

4. Discussion

We conducted personal interviews with questionnaires, and the targets were mainly software practitioners in different industries. A total of 44 questionnaires from software practitioners were collected. Table 5 displays the F-test and *p*-value to show that all research hypotheses are supported. This demonstrates that Product Management (PM) of SPLE-Scrum has a significant and beneficial effect on Project Success (PS) (f = 9.457, $p = 0.004$). Project Requirements (PR) of SPLE-Scrum significantly positively affect PS (f = 16.286, $p = 0$). Product Architecture (PA) of SPLE-Scrum significantly positively affects PS (f = 9.045, $p = 0.005$). By conducting the F-test, it reached a statistically significant level, indicating that the SPLE-Scrum approach has a significant positive impact on project success.

Table 5. Relations between SPLE-Scrum key factors and project success.

Research Hypothesis	R	R-Squared	Adjusted R-Squared	Estimated Standard Error	F-Test	*p*-Value
H1: PM → PS	0.437	0.191	0.171	0.984	9.457	0.004 **
H2: PR → PS	0.538	0.289	0.272	0.923	16.286	0.000 ***
H3: PA → PS	0.429	0.184	0.164	0.988	9.045	0.005 **

Note: (1) *** means $p < 0.001$; (2) ** means $p < 0.01$.

Table 6 shows the comparison table of SPLE-Scrum and the extant literature [29,41]. In the comparison table, "Yes" indicates that the respective approach exhibits the benefit, and

"N/A" denotes that the specific literature does not provide sufficient information related to the corresponding benefit. SPLE-Scrum has contributed to the enhancement of Traditional Scrum and SPLE in the following ways:

(1) SPLE-Scrum uses Scrum as a basis to include SPLE activities to develop software product families, while AgiFPL [41] includes Scrum activities in SPLE. Since SPLE-Scrum is based on Scrum, which software companies widely adopt, it provides a familiar and more straightforward path for software companies to adopt SPEL-Scrum.
(2) SPLE-Scrum introduces the product management meeting to discuss software reuse by initially establishing the product roadmap, variability, and backlog items and updating these roadmap, variability, and backlog items as necessary periodically.
(3) SPLE-Scrum analyzes the domain and application's feature and variability model with user stories and context matrix in the Pre-Sprint. It emphasizes the implementation of core assets using the use-case model and component model during the Sprint for the product line.

Table 6. Comparison with related work.

Approach	**SPLE-Scrum**	**AgiFPL [41]**	**Targeted Scrum [29]**
Agile Adoption	Yes	Yes	Yes
SPLE Adoption	Yes	Yes	N/A
Process Basis	Scrum	SPLE	Scrum
Reusability	Yes	Yes	N/A
Modeling Examples	Feature Model, User Story, Use Case Model, Component Model	Goal Model, Feature Model	Line of Effort
Methodology Focus	Hybrid approach integrating SPLE and Scrum focuses on agility, reusability, requirements analysis, product architecture, and product management.	Agile Product Line Engineering method focuses on managing software product lines with agility.	Agile methodology with a focus on mission command principles

Overall, SPLE-Scrum demonstrates its benefits by enhancing Traditional Scrum, including SPLE activities within the Scrum framework, providing a basis for easier adoption, introducing the Product Management meeting, and emphasizing the integration of domain and application features in software product family development. These contributions make SPLE-Scrum a valuable and practical approach for organizations seeking to optimize their software engineering practices and incorporate agile development and software reuse capabilities.

5. Conclusions

This study provides a hybrid agile development approach to include the management and engineering practices of SPLE and Scrum. In the Pre-Sprint, the product requirements are addressed by domain and application requirements engineering. The domain and application design generates a reference architecture with reusable components during the Sprint. Developers can make specific choices without designing an application architecture from scratch but make specific choices by linking the variability derived from the reference architecture. The core assets are reused to build a series of applications with various product lines. By leveraging the strengths of both approaches, SPLE-Scrum can improve software reusability in software development while reducing overall development effort and cost, thereby contributing to project success.

The limitation of this study is the focus on a specific case study involving a blockchain online store. While the case study demonstrates the practical application and benefits of the SPLE-Scrum approach in that particular context, it may limit the generalizability of the findings to other industries and software development projects. The unique characteristics and requirements of the blockchain online store may not fully represent the challenges and dynamics in different domains.

Therefore, further work could explore the application of SPLE-Scrum in different industries, such as healthcare, finance, or automotive, to investigate its effectiveness in addressing specific challenges within those domains. Additionally, exploring the scalability of the SPLE-Scrum approach for large-scale and complex software development projects would be beneficial for understanding its potential impact in broader organizational settings. Furthermore, investigating the incorporation of emerging technologies and development practices, such as DevOps or continuous integration/continuous delivery (CI/CD), within the SPLE-Scrum approach could enhance its adaptability and efficiency in the fast-paced and rapidly evolving software development landscape. Additionally, investigating the impact of team size and composition on the successful adoption of SPLE-Scrum would offer valuable insights into the approach's suitability for different team structures and dynamics.

In summary, while this study presents a promising hybrid agile development approach in SPLE-Scrum, there is scope for future research to enhance its applicability, generalizability, and efficiency in various software development contexts. Expanding the scope of case studies, exploring the integration of emerging technologies, and evaluating the impact of team dynamics are some potential directions to further develop and validate the potential of SPLE-Scrum in advancing software engineering practices.

Author Contributions: Conceptualization, W.-T.L. and C.-H.C.; Methodology, W.-T.L. and C.-H.C.; Validation, C.-H.C. and W.-T.L.; Writing—original draft, W.-T.L. and C.-H.C.; Writing—review & editing, W.-T.L.; Supervision, W.-T.L.; Project administration, W.-T.L.; Funding acquisition, W.-T.L. Validation, C.-H.C. and W.-T.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Technology in Taiwan under grants MOST 111-2221-E-017-003.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sutherland, J. *Scrum: The Art of Doing Twice the Work in Half the Time*; Crown Business: New York, NY, USA, 2014.
2. Williams, L. Agile Software Development Methodologies and Practices. *Adv. Comput.* **2010**, *80*, 1–44.
3. Almeida, F.; Espinheira, E. Adoption of Large-Scale Scrum Practices through the Use of Management 3.0. *Informatics* **2022**, *9*, 20. [CrossRef]
4. Edison, H.; Wang, X.; Conboy, K. Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review. *IEEE Trans. Softw. Eng.* **2022**, *48*, 2709–2731. [CrossRef]
5. Spagnoletti, P.; Kazemargi, N.; Prencipe, A. Agile Practices and Organizational Agility in Software Ecosystems. *IEEE Trans. Eng. Manag.* **2022**, *69*, 3604–3617. [CrossRef]
6. Wessel, R.M.v.; Kroon, P.; Vries, H.J.d. Scaling Agile Company-Wide: The Organizational Challenge of Combining Agile-Scaling Frameworks and Enterprise Architecture in Service Companies. *IEEE Trans. Eng. Manag.* **2022**, *69*, 3489–3502. [CrossRef]
7. Sherif, E.; Helmy, W.; Galal-Edeen, G.H. Proposed Framework to Manage Non-Functional Requirements in Agile. *IEEE Access* **2023**, *11*, 53995–54005. [CrossRef]
8. Pohl, K.; Böckle, G.; van Der Linden, F.J. *Software Product Line Engineering: Foundations, Principles and Techniques*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2005.
9. Böckle, G.; Pohl, K.; van der Linden, F. A framework for software product line engineering. In *Software Product Line Engineering*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 19–38.
10. Weiss, D.M.; Lai, C.T.R. *Software Product-Line Engineering: A Family-Based Software Development Process*; Addison-Wesley Reading: Boston, MA, USA, 1999; Volume 12.

11. Northrop, L.; Clements, P.; Bachmann, F.; Bergey, J.; Chastek, G.; Cohen, S.; Donohoe, P.; Jones, L.; Krut, R.; Little, R. *A Framework for Software Product Line Practice, Version 5.0*; Software Engineering Institute | Carnegie Mellon University: Pittsburgh, PA, USA, 2012.
12. Moon, M.; Yeom, K.; Chae, H.S. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Trans. Softw. Eng.* **2005**, *31*, 551–569. [CrossRef]
13. Pine, B.J.; Pine, J.; Pine, B.J.I. *Mass Customization: The New Frontier in Business Competition*; Harvard Business Press: Brighton, MA, USA, 1993.
14. Jabar, M.A.; Abdullah, S.; Jusoh, Y.Y.; Mohanarajah, S.; Ali, N.M. Adaptive and Dynamic Characteristics in Hybrid Agile Management Model for Software Development Project Success. In Proceedings of the 2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS), Johor Bahru, Malaysia, 2–3 December 2019; pp. 1–5. [CrossRef]
15. Wan, J.; Zhu, Y.; Zeng, M. Case study on critical success factors of running Scrum. *J. Softw. Eng. Appl.* **2013**, *6*, 59–64. [CrossRef]
16. Highsmith, J.; Cockburn, A. Agile software development: The business of innovation. *Computer* **2001**, *34*, 120–127. [CrossRef]
17. Anderson, D.J. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*; Prentice Hall: Upper Saddle River, NJ, USA, 2003.
18. Hoda, R.; Salleh, N.; Grundy, J.; Tee, H.M. Systematic literature reviews in agile software development: A tertiary study. *Inf. Softw. Technol.* **2017**, *85*, 60–70. [CrossRef]
19. Martin, J. *Application Development without Programmers*; Prentice Hall PTR: Hoboken, NJ, USA, 1982.
20. Hoda, R.; Salleh, N.; Grundy, J. The Rise and Evolution of Agile Software Development. *IEEE Softw.* **2018**, *35*, 58–63. [CrossRef]
21. Cockburn, A. *Agile Software Development*; Addison-Wesley Longman: Boston, MA, USA, 2002; pp. 1–221.
22. VersionOne, C. *12th Annual State of Agile Report*; CollabNet VersionOne. Com: Alpharetta, GA, USA, 2018.
23. Takeuchi, H.; Nonaka, I. The new new product development game. *Harv. Bus. Rev.* **1986**, *64*, 137–146.
24. Schwaber, K.; Beedle, M. *Agile Software Development with SCRUM*; Prentice Hall: Upper Saddle River, NJ, USA, 2002; Volume 1.
25. Schwaber, K.; Sutherland, J. *The Scrum GuideTM. The Definitive Guide to Scrum: The Rules of the Game*. 2020. Available online: https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf (accessed on 1 February 2023).
26. Blankenship, J.; Bussa, M.; Millett, S. Managing Agile Projects with Scrum. In *Pro Agile .NET Development with Scrum*; Apress: Berkeley, CA, USA, 2011. [CrossRef]
27. Rubin, K.S. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*; Addison-Wesley: Boston, MA, USA, 2012.
28. Larsen, D.; Derby, E. *Agile Retrospectives*; Pragmatic Bookshelf: Raleigh, NC, USA, 2006.
29. Harvie, D.P.; Agah, A. Targeted scrum: Applying mission command to agile software development. *IEEE Trans. Softw. Eng.* **2016**, *42*, 476–489. [CrossRef]
30. Wisocky, R.K. *Effective Project Management: Traditional, Adaptive, Extreme*; Wiley: Indianapolis, IN, USA, 2007.
31. Clements, P.; Northrop, L. *Software Product Lines: Practices and Patterns*; Addison-Wesley Professional: Boston, MA, USA, 2001.
32. Khan, S.A.; Alenezi, M.; Agrawal, A.; Kumar, R.; Khan, R.A. Evaluating Performance of Software Durability through an Integrated Fuzzy-Based Symmetrical Method of ANP and TOPSIS. *Symmetry* **2020**, *12*, 493. [CrossRef]
33. Dospinescu, O.; Perca, M. *Technological Integration for Increasing The Contextual Level of Information*; Editura Universitatii Alexandru Ioan Cuza Iasi: Iasi, Romania, 2011.
34. Ling, Y.; An, T.; Yap, L.W.; Zhu, B.; Gong, S.; Cheng, W. Disruptive, Soft, Wearable Sensors. *Adv. Mater.* **2020**, *32*, 1904664. [CrossRef]
35. Aguado, A.; Lopez, V.; Lopez, D.; Peev, M.; Poppe, A.; Pastor, A.; Folgueira, J.; Martin, V. The Engineering of Software-Defined Quantum Key Distribution Networks. *IEEE Commun. Mag.* **2019**, *57*, 20–26. [CrossRef]
36. Buraga, S.C.; Amariei, D.; Dospinescu, O. An OWL-Based Specification of Database Management Systems. *Comput. Mater. Contin.* **2022**, *70*, 5537–5550. [CrossRef]
37. Lee, W.-T.; Ma, S.-P. Process modeling and analysis of service-oriented architecture–based wireless sensor network applications using multiple-domain matrix. *Int. J. Distrib. Sens. Netw.* **2016**, *12*, 1550147716676556. [CrossRef]
38. Santos, A., Jr.; de Lucena, V.F., Jr. ScrumPL-Software Product Line Engineering with Scrum. In Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, Greece, 22–24 July 2010; pp. 239–244.
39. Tian, K.; Cooper, K. Agile and software product line methods: Are they so different. In Proceedings of the 1st International Workshop on Agile Product Line Engineering, Baltimore, MD, USA, 21 August 2006.
40. Díaz, J.; Pérez, J.; Alarcón, P.P.; Garbajosa, J. Agile product line engineering—A systematic literature review. *Softw. Pract. Exp.* **2011**, *41*, 921–941. [CrossRef]
41. Haidar, H.; Kolp, M.; Wautelet, Y. Agile Product Line Engineering: The AgiFPL Method. In Proceedings of the 12th International Conference on Software and Data Technologies, Madrid, Spain, 24–26 July 2017; pp. 275–285.
42. Noor, M.A.; Rabiser, R.; Grünbacher, P. Agile product line planning: A collaborative approach and a case study. *J. Syst. Softw.* **2008**, *81*, 868–882. [CrossRef]
43. Metzger, A.; Pohl, K. Software product line engineering and variability management: Achievements and challenges. In *Future of Software Engineering Proceedings (FOSE 2014)*; Association for Computing Machinery: New York, NY, USA, 2014; pp. 70–84.
44. Takpuie, D.; Tanner, M. Investigating the characteristics needed by scrum team members to successfully transfer tacit knowledge during agile software projects. *Electron. J. Inf. Syst. Eval.* **2016**, *19*, 36–54.

45. Kalbach, J.; Kalbach, J. *Mapping Experiences: A Guide to Creating Value through Journeys, Blueprints, and Diagrams*; O'Reilly: Sebastopol, CA, USA, 2016.
46. Patton, J.; Economy, P. *User Story Mapping: Discover the Whole Story, Build the Right Product*, 1st ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2014.
47. Cohn, M. *User Stories Applied: For Agile Software Development*; Addison Wesley Longman Publishing Co., Inc.: Upper Saddle River, NJ, USA, 2004.
48. Beck, K. Extreme Programming. In Proceedings of the Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 29 (Cat. No. PR00275), Nancy, France, 7–10 June 1999; p. 411.
49. Ambler, S. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*; John Wiley & Sons: New York, NY, USA, 2002.
50. Fernández-Diego, M.; Méndez, E.R.; González-Ladrón-De-Guevara, F.; Abrahão, S.; Insfran, E. An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review. *IEEE Access* **2020**, *8*, 166768–166800. [CrossRef]
51. Khudhair Abbas, A.; Hassan Safi, H.; Qasim AlBawi, S. Using Feature and Orthogonal Variability Models to Design E-Commerce Model With (Software Product Line Engineering) technique. *J. Kerbala Univ.* **2017**, *13*, 169–178.
52. Beck, K. Embracing change with extreme programming. *Computer* **1999**, *32*, 70–77. [CrossRef]
53. Anwer, F.; Aftab, S.; Shah, S.M.; Waheed, U. Comparative Analysis of Two Popular Agile Process Models: Extreme Programming and Scrum. *Int. J. Comput. Sci. Telecommun.* **2017**, *8*, 1–7.