



Software Product Line Engineering in Supply Chain Management Systems for Manufacturing Sector

Jehian Norman Saviero¹, Muhammad Raihan², Oman Komarudin³, Ade Azurat⁴

^{1,3,4}Faculty of Computer Science, Universitas Indonesia, Depok, Indonesia

²Department of Financial Technology, Traveloka, Tangerang, Indonesia

¹jehian.norman01@ui.ac.id, ²muhammad.raihan@traveloka.com, ³oman.komarudin@ui.ac.id, ⁴ade.azurat@cs.ui.ac.id

Abstract

Manufacturing companies are industrial enterprises that process raw materials and implement Supply Chain Management (SCM). SCM encompasses three stages: material management, planning and control, and production. While these stages are common across manufacturing companies, the workflows and strategies employed vary based on the type of goods produced. For example, one company typically approaches process orders based on requests, whereas the other processes orders based on stock availability. To address these similarities and differences, a software product line engineering (SPLE) approach can be utilized to develop SCM systems. This approach has already been proven effective in other cases, such as developing various product specifications for our Crowdfunding Application (Amanah CS UI) partner. SPLE follows the principle of mass customization, analyzing the commonalities and variabilities of the SCM system to meet diverse company needs. This approach improves the cost optimization and time efficiency in developing various SCM specifications to fulfill the requirements of each company. The development of the SCM system in this study adopts a delta-oriented programming paradigm and Abstract Behavioral Specification programming language. Subsequently, a comparison was made between the development of the SCM system using the SPLE approach and the clone-and-own approach. The research results in an enhanced SCM system developed through the SPLE, establishing it as the primary solution to existing development issues: reusing shared components and adding new custom components. Additionally, it includes an analysis that compares the SPLE approach with the clone-and-own method.

Keywords: delta-oriented programming; software product line engineering; supply chain management

How to Cite: J. N. Saviero, M. Raihan, O. Komarudin, and A. Azurat, "Software Product Line Engineering in Supply Chain Management Systems for Manufacturing Sector", *J. RESTI (Rekayasa Sist. Teknol. Inf.)*, vol. 9, no. 5, pp. 1002 - 1012, Oct. 2025.

Permalink/DOI: <https://doi.org/10.29207/resti.v9i5.5605>

Received: November 11, 2023

Accepted: April 6, 2025

Available Online: October 3, 2025

This is an open-access article under the CC BY 4.0 License

Published by Ikatan Ahli Informatika Indonesia

1. Introduction

Manufacturing companies are industrial enterprises that engage in the processing of raw materials into finished or semi-finished goods. These companies typically implement Supply Chain Management (SCM) practices, which encompass planning, control, product implementation, and distribution. The production process of a manufacturing company generally involves three stages: material management, planning and control, and production process [1]. In recent decades, manufacturing companies have leveraged information technology to optimize their SCM processes, aiming to minimize costs while enhancing speed and reliability [2]. The use of these technologies enables companies to swiftly address market needs, save operating expenses, and enhance overall supply chain agility and resilience [3].

While the overall manufacturing process is similar across companies, each manufacturing company has unique requirements and characteristics for its SCM systems. For instance, although there are standardization efforts by Standard Development Organizations regarding production planning processes, alternative workflows exist. These alternatives include strategies to fulfill product demand, such as make-to-stock (based on product availability) or make-to-order (based on specific customer orders) [4].

Make-to-stock produces goods in advance based on demand forecasts, ensuring quick availability but risking overproduction. Make-to-order, in contrast, minimizes inventory by starting production only after receiving an order, reducing storage costs but leading to longer wait times. Choosing the right strategy is key to balancing costs, speed, and supply chain efficiency

while meeting customer expectations [5]. Meeting the SCM system needs of diverse manufacturing companies using traditional approaches can be costly due to the customization required for each system. Additionally, developing systems with different specifications from scratch is time-consuming. Software Product Line Engineering (SPLE) offers a cost-effective approach to tailoring software products by utilizing shared components and services across different types of software products. SPLE involves the development of software using a platform and mass customization, where the platform serves as a foundation for building other technologies, and mass customization refers to large-scale production customized to individual customer preferences [6].

The SPLE approach enables the development of SCM systems with various specifications to meet the requirements of different companies while reducing costs and effort compared to traditional approaches. In SPLE, features common to all types of software products and features specific to certain types are categorized separately. Development focuses on the common features, which can be further modified while managing the variability features. This approach allows developers to reuse features considered reusable [7].

One specific technique within SPLE is the use of the Abstract Behavioral Specification (ABS) approach, which emphasizes modeling software behavior at a high level of abstraction. ABS provides a formal and concise way to describe the essential features and interactions of a software system. In the context of SPLE, ABS supports delta-oriented programming, a paradigm that facilitates the systematic handling of variations across product variants [8]. Delta-oriented programming involves defining and managing changes, or "deltas," between different products in a product line. ABS enables the specification of these deltas in a clear and modular manner, allowing developers to express the differences between various products without duplicating code unnecessarily. The framework that can support ABS was already developed [9] and was named an **ABS**. To avoid any confusion between the framework name and one of the SPLE technique's names, in this study the framework will be mentioned as ABS Framework. ABS Framework itself is also commonly used as the core framework for ABS [10].

This research aims to address the integration of material planning and production processes to develop a comprehensive SCM system. Additionally, variations of SCM systems will be created through the SPLE approach with the delta-oriented programming paradigm, and a comparison will be made with the clone-and-own method.

2. Methods

A product line refers to a collection of products from a manufacturer (company or individual) that exhibit

certain uniformity and ideally consist of reusable components [11]. Typically, a product line is created to fulfill a specific objective or cater to a particular market segment [12]. SPLE is a method for software development that follows the principles of Platform and Mass Customization. The Platform principle refers to a technological foundation that can be used as a basis for developing other technologies, while Mass Customization involves large-scale production of goods that meet individual user needs [6].

It can be observed from Figure 1 that using the SPLE method involves an upfront investment, meaning that the initial development cost is higher compared to the clone-and-own approach. However, as the number of product variations developed increases, the cost of developing these variations decreases until it reaches a break-even point.

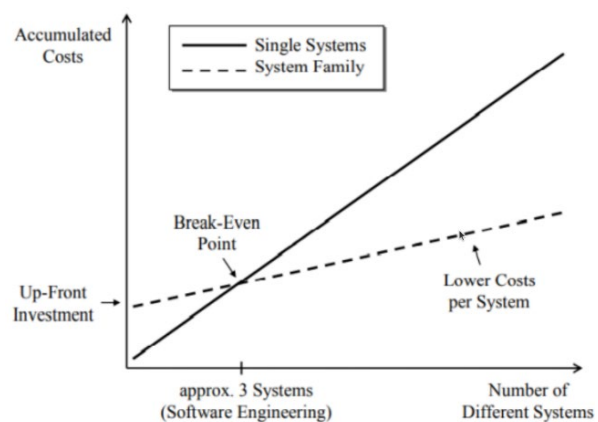


Figure 1. Cost for developing N system variation [6].

Beyond this point, the cost required to create n variations using the SPLE method is lower than the traditional clone-and-own approach.

The process of delivering a finished product to the consumer is a lengthy one, starting from the supplier providing raw materials, going through the production process, and ultimately reaching the consumer through distribution. The supply chain encompasses all activities and processes involved in delivering a product or service to the consumer. Broadly, it consists of three stages: physical supply, manufacturing planning control, and physical distribution. This research focuses on the manufacturing planning-control stage, specifically the material management, planning and control, and production process. As computer and technology advancements continue, companies are developing systems to optimize the supply chain process, aiming to maximize the flow of materials, information exchange between divisions, and the flow of funds [13].

The research utilized the Action Research methodology, where active participation was involved to gain a comprehensive understanding and optimize the process. The stages included Domain Requirement

Engineering, Domain Design, and Domain Realization [6]. Prior to commencing the research, an exploration of the Software Product Line Engineering approach for software development took place. Subsequently, knowledge was enhanced to achieve research objectives. During the preparation phase, all requirements for the SCM application were defined, including domain requirement engineering.

In the design stage, the integration of various feature models from previous research [14], [15] resulted in a comprehensive feature model for the SCM system, covering material management, planning and control, and production processes. A reference architecture document was developed as a guide for implementing reusable artifacts using code, represented by a detailed UML-DOP annotated diagram to depict the system [16].

In the realization stage, module integration and product configuration were carried out to generate a fully functional SCM application that serves in Microservices that aligned with SPLE [17] using ABS Microservice Framework [18] which had previously been employed and instantiated in non-SCM contexts [19]. Adjustments were made to modules to avoid conflicts between interrelated implementations from

different research studies, based on the combined feature model. Additionally, solutions to known software development issues were identified using the ABS Framework.

Following the realization process, testing was conducted to ensure the desired functionality of the SCM system developed through the SPLE method. Finally, submodules of the SCM system were developed using the clone-and-own approach to compare development processes, considering aspects such as effort, potential issues, frameworks, and other relevant factors.

3. Results and Discussions

In the initial stage of this research, a design phase was conducted, resulting in a feature model calibrated from previous research. Integration and merging of the feature models were necessary to ensure artifact reuse without errors or conflicts. The integrated feature model for this case is shown in Figure 2. The integrated feature model was represented in a reference document using UML-DOP notation, including package diagrams, component diagrams, and class diagrams.

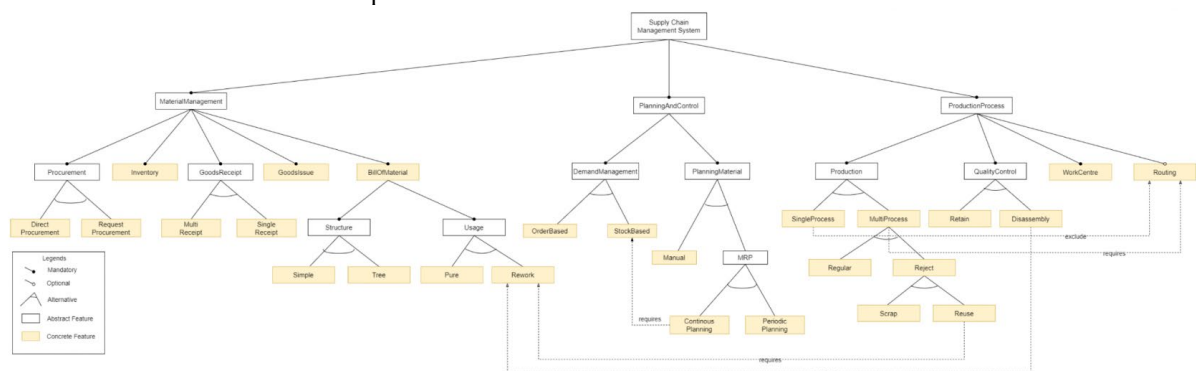


Figure 2. The integrated feature models.

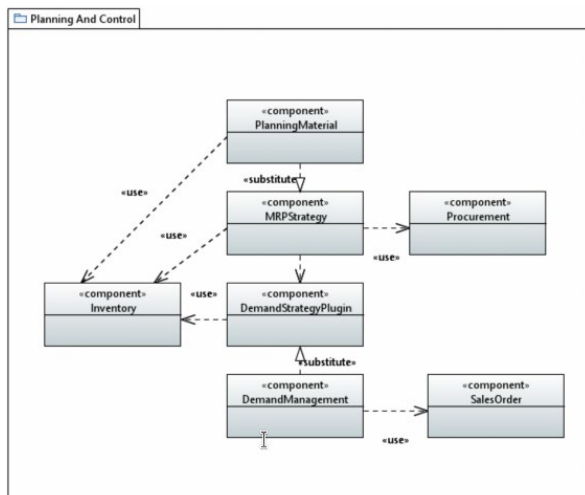


Figure 3. Component Diagram of Planning and Control Process.

For instance, in the component diagram shown in Figure 3, the main components are planning material and demand management, connected to the sales order component. When the MRP feature is selected, the MRP strategy component can be utilized. Additionally, if the stock-based feature is chosen for company strategy, the demand strategy plugin will be utilized.

After the design phase, module integration between the planning material and demand material was performed to satisfy requirements that were already stated in the integrated feature model in Figure 2.

The integration mentioned combines the feature into a single ABS Code. After fulfilling all the requirements, the ABS Code meets the requirements shown in Algorithm 1.

Algorithm 1: MasterMaterialResource

```

1  Function: list(request: HttpRequest) →
2  List<MasterMaterial>
3  Function: save(request: HttpRequest) →
4  List<MasterMaterial>
5  Function: update(request: HttpRequest) →
6  MasterMaterial
7  Function: detail(request: HttpRequest) →
8  MasterMaterial
9  Function: delete(request: HttpRequest) →
10 List<MasterMaterial>
11 Function: findById(materialId: Int) →
12 MasterMaterial
13 Function: findByCode(materialCode:
14 String) → MasterMaterial
15 Function: getMaterialType(id: Int) →
16 String
17 Function: getByType(materialType: String)
18 → List<MasterMaterial>
19 Function:
20 transferReservedStockToAdd(masterMaterial:
21 MasterMaterial, quantity: Int) → Int
22 Function:
23 transferReserverStockToReduce(masterMaterial:
24 MasterMaterial, quantity: Int) → Int
25 Function: reduceStock(masterMaterial:
26 MasterMaterial, quantity: Int) → Unit
27 Function: addStock(masterMaterial:
28 MasterMaterial, quantity: Int) → Unit
29 Function:
30 receiptStockFromReserved(materialCode:
31 String, receiptQuantity: Int) → Unit

```

Algorithm 1. Functional requirement of MasterMaterialResource after do integration.

Method `list` (line 1 at Algorithm 1) is used to list down all of purchase requisitions without using any filtering method. The method `save` on the other hand will do insertion object of `MasterMaterial` into Database.

Following module integration, the product configuration was adjusted to create a functional and complete SCM system. The selected features for the SCM system were a combination of those generated in previous research [14]. Changes and adjustments were made to ensure the smooth flow of features without disruptions. Figure 4 illustrates a snippet of the adjustments made to the `ProductConfig` requirement specification. The previous research only provides functional requirements in the section `Features` and this paper will implement the section `Delta Modules` to support new variants.

After successful module integration, alternative solutions were identified for development issues. One such issue was the absence of an authentication feature in the SCM application. Based on the previous research [20], authentication and access control were implemented. Authentication utilized a third-party single sign-on (SSO) provider, while access control was defined by each service

In the development of software using the SPLE method and ABS Microservices Framework, a known issue pertains to foreign keys [18]. The version of ABS

Microservices utilized in this study lacks support for a model to store attributes in the form of foreign keys that are linked to objects from other models stored in the database. The use of foreign keys in the model is crucial, particularly when an application employs a relational database like the SCM system developed in this study, which utilizes PostgreSQL as the database. Additionally, the utilization of foreign keys in microservices is significant as there are interconnections between services, and there may also be relationships between models within these services. To address the existing foreign key issue in the ABS Microservices Framework, several alternative solutions were implemented, including: Storing the primary key of the related model's rows; Creating a new class with a placeholder for the other model; Creating a view from the database perspective.

Configuration: SCM Product Line**Features:**

- Material Management
- Goods Issue
- Goods Receipt
- Receipt Types: Single, Multiple
- Warehouse Management
- Procurement
 - Direct Procurement
 - Request Procurement
- Bill of Material (BOM)
 - Structure Types: Simple, Tree
 - Usage Types: Pure, Rework
- Planning and Control
 - Material Requirements Planning (MRP)
 - Planning Types: Continuous,

Periodic

- Demand Management
- Order-Based Planning
- Stock-Based Planning
- Production Process
 - Single Process
 - Multi-Process
 - Regular Production
 - Reject and Scrap Handling
 - Reuse
 - Quality Control
 - Retain
 - Disassembly
- Work Centre
- Routing

Delta Modules:

- Rework Variant Model
- Rework Variant Resource
- Rework Bill of Material Model
- Rework Bill of Material Resource
- Rework Material Process Model
- Rework Material Process Resource

Figure 4. ProductConfig Specification.

3.1 Storing the primary key of the related models' row

The basic idea of this approach is creating an attribute that stores another Id of another model that represents another table's primary key that is related with the current Model that represents the current table's primary key. Let's say the Model named `OrderDetail` has a

one-to-one or one-to-many relationship with **MasterMaterial** that is described in Algorithm 2 for Model **OrderDetail** and Algorithm 4 for Model **MasterMaterial**. We can take a look at lines 3-4 of the functional requirement of model **OrderDetail** at Algorithm 2 and its algorithm implementation of **OrderDetailImpl** at Algorithm 3 especially in lines 3, 8-9, and 10-12 to give a better picture of how the implementation is done. Those lines will be used as a reference for the primary key of Model **MasterMaterial** that is described in Algorithm 4 at line 2 and its algorithm implementation of **MasterMaterialImpl** in Algorithm 5 at line 2.

Algorithm 2: OrderDetail

```

1  Function: getId() → Int
2  Function: setId(id: Int) → Unit
3  Function: getMaterialId() → Int
4  Function: setMaterialId(materialId:
5  Int) → Unit

```

Algorithm 2. Functional Requirement of **OrderDetail** Model.

Algorithm 3: OrderDetailImpl

```

1  Data:
2      id: Int // PK of OrderDetail
3      materialId: Int // Foreign Key
4  Function: getId() → Int
5      Return: id
6  Function: setId(id: Int) → Unit
7      Set: this.id ← id
8  Function: getMaterialId() → Int
9      Return: materialId
10 Function: setMaterialId(materialId:
11 Int) → Unit
12 Set: this.materialId ← materialId

```

Algorithm 3. Algorithm Implementation of **OrderDetail** Model.

Algorithm 4: MasterMaterial

```

1  Function: getId() → Int
2  Function: setId(id: Int) → Unit
3  Function: getCode() → String
4  Function: setCode(code: String) → Unit

```

Algorithm 4. Functional Requirement of **MasterMaterial** Model.

Algorithm 5: MasterMaterialImpl

```

1  Data:
2      id: Int // PK of MasterMaterial
3      code: String
4  Function: getId() → Int
5      Return: id
6  Function: setId(id: Int) → Unit
7      Set: this.id ← id
8  Function: getCode() → String
9      Return: code
10 Function: setCode(code: String) → Unit
11 Set: this.code ← code

```

Algorithm 5. Algorithm Implementation of **MasterMaterial** Model.

Storing the **materialId** will make the relation between Model **OrderDetail** with Model **MasterMaterial** can be established. Even the relationship can be established, retrieving both model should be applied separately on the *frontend* side. For this case, we retrieving the **OrderDetail** first. Afterwards, based on the **materialId** on the

OrderDetail, we retrieve the **MasterMaterial**. Currently, there is no join-sql-like operation to fulfill this approach.

3.2 Creating a new class with a placeholder for the other model.

The approach that implemented to achieve this is storing of the other model details on the class that needs establish relationship between two tables. Let's say Model **BillofMaterial** have one-to-one relationship with Model **MasterMaterial**. The details of Model **BillofMaterial** described in Algorithm 6 and its algorithm implementation of **BillofMaterialImpl** at Algorithm 7.

Algorithm 6: BillofMaterial

```

1  Function: getId() → Int
2  Function: setId(id: Int) → Unit
3  Function: getMasterMaterial() →
4  Map<String, String>
5  Function:
6  setMasterMaterial(masterMaterial:
7  MasterMaterial) → Unit

```

Algorithm 6. Functional Requirement of **BillofMaterial** Model.

Algorithm 7: BillofMaterialImpl

```

1  Data:
2      id: Int // PK of BillofMaterial
3      masterMaterial: Map<String, String>
4  Function: getId() → Int
5      Return: id
6  Function: setId(id: Int) → Unit
7      Set: this.id ← id
8  Function: getMasterMaterial() →
9  Map<String, String>
10 Return: masterMaterial
11 Function:
12 setMasterMaterial(masterMaterial:
13 MasterMaterial) → Unit
14 Data: newMasterMaterial ← map[]
15 newMasterMaterial ←
16 insert(newMasterMaterial, Pair("code",
17 masterMaterial.getCode()))
18 Set: this.masterMaterial ←
19 newMasterMaterial

```

Algorithm 7. Algorithm Implementation of **BillofMaterial** Model.

The **BillofMaterial** will storing the detail of **MasterMaterial** that corresponded with the selected **BillofMaterial**. From *frontend* side, they need to call backend service once but still backend service still calls the database twice to retrieve each model.

3.3 Creating a view from the database perspective

A view in database terminology refers to a query that can be an alternative for displaying various data from tables. Each view is defined from one or more base tables. By using views, queries involving multiple tables that previously required JOIN can be simplified. Views eliminate the need for JOIN. In essence, a view assigns a name to a query, commonly used for complex and frequently used queries.

The usage of views serves as an alternative approach to address foreign key issues. With views, the utilization of foreign keys can be defined on the database side by only storing primary keys, similar to the initial solution. Algorithm 8 is an example query that can be used to retrieve information on `BillofMaterial` data along with associated `MasterMaterial` data.

Algorithm 8: RetrieveBillofMaterialDetails

```

1  Result: result_set
2  Query: SELECT b.id, b.subQuantity,
3  mm.id AS material_id, b.materialCode AS
4  material_code,
5  mm.name AS material_name,
6  mm.stock AS material_stock,
7  mm.reservedStock AS material_res_stock,
8  mm.uom AS material_uom,
9  mm.materialType AS material_type,
10 mm.scheduleTime AS
11 material_sched_time, mm.planningTimeFence
12 AS material_ptf,
13 mm.procurementType AS
14 material_proc_type, mm.supplierId AS
15 material_supp_id
16 FROM billofmaterialimpl AS b
17 JOIN mastermaterialimpl AS mm ON
18 b.materialcode = mm.code;
19
20 // Process the result set (not shown in
21 this pseudocode)
22 ProcessResultSet(result_set);

```

Algorithm 8. Algorithm create view of between Model BillofMaterial and Model MasterMaterial.

In query at Algorithm 8, JOIN is used with the condition that the `materialCode` from the Model `BillofMaterial` table matches the code from the Model `MasterMaterial` table. To streamline this query, the view was based on the query named `BoMMMview`, allowing the above query to be executed without using JOIN. By eliminating JOIN, the query used to obtain information from various tables in the database becomes significantly shorter. However, the drawback of this solution is that the changes are not made in the application code itself but rather serve as a workaround to optimize information retrieval. This method enables the *frontend* application to directly fetch data from the database, bypassing the developed backend application. Prior to conducting testing, the product was defined in the ABS programming language using the product selection language. In the product selection language, a product is defined by specifying its included features. Algorithm 9 is a complete example of how to achieve defining two products, `OrderBasedSimple` and `StockBasedSimple` with different features that listed down in Figure 4 previously. Based on that Algorithm, we know that when we want to re-use some of the features and wants to change the specific feature, we only change the feature with its respected value. For this case, the 7th parameter on the SELECT query on view table `BoMMMview` (`material_res_stock`), product

`OrderBasedSimple` will use `OrderBased` and for product `StockBasedSimple` will use `StockBased`.

Algorithm 9: ProductSelection

```

1  // Define the attributes for product
2  "OrderBasedSimple"
3  Product: OrderBasedSimple
4  Features: Simple, Pure, Warehouse,
5  DirectProcurement, SingleReceipt,
6  GoodsIssue, OrderBased, PlanningMaterial,
7  SingleProcess,
8  Retain, WorkCentre
9
10 // Define the attributes for product
11 "StockBasedSimple"
12 Product: StockBasedSimple
13 Features: Simple, Pure, Warehouse,
14 DirectProcurement, SingleReceipt,
15 GoodsIssue, StockBased, PlanningMaterial,
16 SingleProcess, Retain, WorkCentre

```

Algorithm 9. Algorithm that defining Product Selection Language.

To provide empirical evidence for the developed alternative solution, the implementation results of the previously described authentication feature will be tested. The API utilized for testing the implementation of the authentication feature is the list API for the bill of material feature. The desired output from invoking this API is a list of bills of material information stored in the database. Postman¹ was employed as the tool for sending requests during the testing phase. To support replicating the testing result, Table 1 showing the sample of API spec that were used on Postman for the testing in this study. After a series of tests were conducted on the implementation of several alternative solutions for the foreign key issue in development, Table 2 presents the test results that were already explained in the previous section. Based on the testing results and analysis of the strengths and weaknesses of each alternative solution, the chosen option is to store the Primary Key of related objects due to the minimal additional effort required to make API calls compared to other additional efforts. After ensuring the successful implementation of the alternative solution for various development issues, further testing was conducted on different SCM system variations that can be generated from the developed program. The testing focused on two main aspects: variations in the data model stored in the database and variations in different business logic flows. To comprehensively test these aspects, a modified frontend application from a manufacturing company's SCM system was utilized. For the data model variation testing, the selected product feature specification focuses on the usage of the bill of materials, specifically with two variations: `simple` and `rework`. In the `simple` usage, the `BillofMaterial` model remains unmodified. Figure 5 shows the `BillofMaterial` page for the `simple` feature and Figure 6 shows the `BillofMaterial` page for `rework` feature.

¹<https://postman.com>. Tools for API development and testing.

Table 1. Postman Testing Scenario

Sample Testing Scenario	API Path	CURL	Parameter	Expected Result	Sample Result
Get a List of Order Details by Adjusting the Order Detail to have materialId as the Foreign Key of the Order Detail model or the Primary Key of other models.	GET /api/orderdetail/list.abs		-	Should return all of the order details with materialId as each order detail's attribute	{ "data": [{ "id": 1, "quantity": 5, "requiredDate": "21/03/2022", "materialId": 15, "createdAt": "14/03/2022", "salesOrderCode": "" }, { "id": 2, "quantity": 3, "requiredDate": "21/03/2022", "materialId": 18, "createdAt": "14/03/2022", "salesOrderCode": "" }] }
Get a Bill of Material by Adjusting the Bill of Material to have an attribute of Map to storing another related model.	GET /api/billofMaterial/de tailForeign.abs		id=15	Should return object Bill of Material with id=15 and has an attribute with type Map that storing another related model.	{ "data": { "id": 15, "materialCode": "WIP001", "subMaterialCode": "WIP002", "subQuantity": 2, "testForeign": "InsertAssoc(Pair(\"relatedMaterialCode\", \"WIP001\"), EmptyMap)" } }

Table 2. Result of Alternative Solution for Foreign Key Issue.

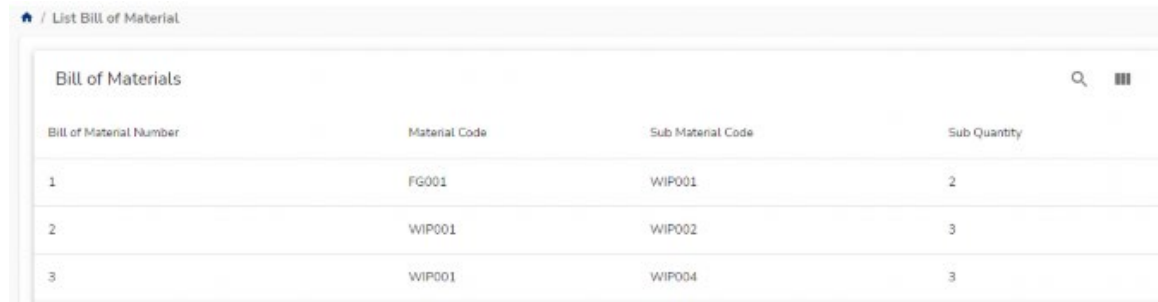
Criteria	Description	Result
FK - Save Primary Key	Storing Primary Key of another model as an attribute	Information about Primary Key from another table was retrieved successfully
FK - Class Placeholder	Creating class that storing another model details and information	Other model details and their information were received. Need additional effort with applying manipulation to achieve this.
FK Database View	- Creating view for database	Data retrieval using View Table successfully run.

Based on the testing results and analysis of the strengths and weaknesses of each alternative solution, the chosen option is to store the Primary Key of related objects due to the minimal additional effort required to make API calls compared to other additional efforts. After ensuring the successful implementation of the alternative solution for various development issues, further testing was conducted on different SCM system variations that can be generated from the developed program. The testing focused on two main aspects: variations in the data model stored in the database and

variations in different business logic flows. To comprehensively test these aspects, a modified frontend application from a manufacturing company's SCM system was utilized. For the data model variation testing, the selected product feature specification focuses on the usage of the bill of materials, specifically with two variations: simple and rework. In the simple usage, the BillOfMaterial model remains unmodified. Figure 5 shows the BillOfMaterial page for the simple feature and Figure 6 shows the BillOfMaterial page for rework feature. Based on the testing results and analysis of the strengths and weaknesses of each alternative solution, the chosen option is to store the Primary Key of related objects due to the minimal additional effort required to make API calls compared to other additional efforts. After ensuring the successful implementation of the alternative solution for various development issues, further testing was conducted on different SCM system variations that can be generated from the developed program. The testing focused on two main aspects: variations in the data model stored in the database and variations in different business logic flows. To comprehensively test these aspects, a modified frontend application from a manufacturing company's SCM

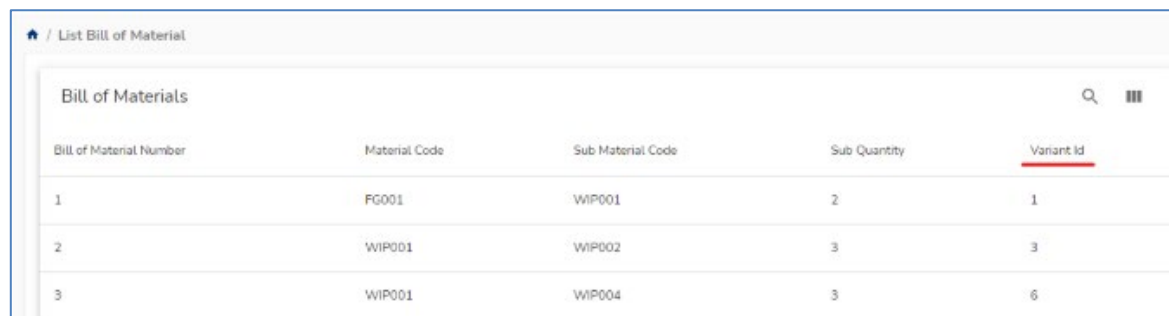
system was utilized. For the data model variation testing, the selected product feature specification focuses on the usage of the bill of materials, specifically with two variations: simple and rework. In the simple usage, the BillOfMaterial model remains unmodified. Figure 5 shows the BillOfMaterial page for the simple feature and Figure 6 shows the BillOfMaterial page for rework feature. Another

important variation is in the application flow, as different manufacturing companies have different production flows and strategies tailored to the products they produce. To illustrate the application flow variations, the generated products include two distinct features in demand management strategy: order-based and stock-based. The differences in the flow between these two variations of the SCM system are described next.



Bill of Material Number	Material Code	Sub Material Code	Sub Quantity
1	FG001	WIP001	2
2	WIP001	WIP002	3
3	WIP001	WIP004	3

Figure 5. Display of BillOfMaterial feature of Simple variation.



Bill of Material Number	Material Code	Sub Material Code	Sub Quantity	Variant Id
1	FG001	WIP001	2	1
2	WIP001	WIP002	3	3
3	WIP001	WIP004	3	6

Figure 6. Display of BillOfMaterial feature of Rework variation.

The key distinction between the two flows lies in the creation of Product Information Repository (PIR). In the order-based approach, adjustments to PIR are made immediately upon receiving an order, regardless of stock availability. However, in stock-based management, PIR adjustments are made only when existing stock is insufficient to fulfill the order. To test these variations, several orders were placed in both systems, yielding the following results.

The testing confirmed that both variations of the SCM system produced the desired outputs. After successful testing of the SCM system and its variations, a comparison was made between the feature variations developed using SPL Engineering and the clone-and-own method. Specifically, a comparison was conducted on the demand management feature, which includes order-based and stock-based variations. The comparison was based on functionality and experience in product development. To implement the clone-and-own approach, Java programming language with the Spring Boot library was used to create a functionally equivalent REST API to the product generated using SPL Engineering.

In the creation of the order-based management system using Spring Boot, various methods were developed to align with the feature flow established through SPL Engineering. An example of such a method is the creation of an order, which invokes the `createPIR` method to adjust the PIR accordingly. On the other hand, the implementation code for order-based management is shown in Algorithm 10 and the implementation code for stock-based management is shown in Algorithm 11 and Algorithm 12.

The example use case of Algorithm 10 `CreateOrderDetailForOrderBased` itself is when the user wants to create order details for the order that will be executed based on the incoming order request without maintaining the current stock available. Basically, when we call `createOrderDetail(5, 2023-12-01, "HONDA1", "0001")` from Algorithm 10 we will directly create an order informing that we need additional material 0001 with quantity 5 to be prepared. On the other hand for Algorithm 11 `CreateOrderDetailForStockBased`, the user case itself is when the user wants to create order details for the order that we also manage the remaining stock that

they have and will be executed with reducing the quantity of remaining stock or adding purchasing additional stock first to fulfill the gap of difference between actual stock with the requested stock. So, when we call `createOrderDetail(3, 2023-12-02, "YAMAHA1", "0002")` from Algorithm 11 we will directly create an order informing that we will reserve material 0002 with quantity 3 on the current stock availability. Algorithm 12 `ReserveStockForStockBased` itself will execute the process to create additional purchasing requests if the quantity needed is greater than the available stock. Otherwise, it'll subtract the remaining stock. At the end of the process, it'll update the reserved stock on the Master Material data.

Algorithm 10: CreateOrderDetailForOrderBased

```

1  Function: createOrderDetail(quantity:
2  Int, requiredDate: LocalDate,
3  salesOrderCode: String, materialId: Int) →
4  OrderDetail
5
6      OrderDetail newOrderDetail ← empty
7  object
8
9      newOrderDetail.quantity ← quantity
10     newOrderDetail.requiredDate ←
11     requiredDate
12     newOrderDetail.salesOrderCode ←
13     salesOrderCode
14
15     MasterMaterial mm =
16     masterMaterialRepository
17     .findById(materialId).get()
18
19     newOrderDetail.masterMaterial ← mm
20     createPIR(newOrderDetail)
21
22     orderDetailRepository.save(newOrderDetail)
23
24     Return: newOrderDetail

```

Algorithm 10. Algorithm of Create Order Detail for Order-Based Management with Clone-and-Own

It is evident that in the creation of an order, the `reserveStock` method is called (line 25 at Algorithm 11) to adjust the stock initially, while the `createPIR` method is only invoked when the order quantity exceeds the available stock.

After conducting the development, a comparison was made by the author across various aspects. The aspects compared are as follows:

Cost of creating application variations. In this study, cost refers to the number of lines of code added. With the SPLE method, developers only need to add product definitions when adding variations. However, with clone-and-own, developers have to duplicate entire modules, resulting in a multiplication of lines of code according to the number of variations.

Maintainability and application updates. Maintainability and updates for product variations are only performed once with the SPLE method, whereas

they need to be repeated multiple times with the traditional approach.

Algorithm 11: CreateOrderDetailForStockBased

```

1  Function: createOrderDetail(quantity:
2  Int, requiredDate: LocalDate,
3  salesOrderCode: String, materialId: Int) →
4  OrderDetail
5
6      OrderDetail newOrderDetail ← empty
7  object
8
9      newOrderDetail.quantity ← quantity
10     newOrderDetail.requiredDate ←
11     requiredDate
12     newOrderDetail.salesOrderCode ←
13     salesOrderCode
14
15     MasterMaterial mm ←
16     masterMaterialRepository
17     .findById(materialId).get()
18
19     newOrderDetail.masterMaterial ← mm
20
21     orderDetailRepository.save(newOrderDetail)
22
23     // Call the reserveStock function
24     with the new OrderDetail
25     reserveStock(newOrderDetail)
26
27     Return: newOrderDetail

```

Algorithm 11. Algorithm of Create Order Detail for Stock-Based Management with Clone-and-Own

Algorithm 12: ReserveStockForStockBased

```

1  Function: reserveStock(orderDetail:
2  OrderDetail) → Void
3
4      MasterMaterial mm ←
5      orderDetail.masterMaterial
6
7      Int: savedStock ← mm.stock
8      Int: quantityNeeded ←
9      orderDetail.quantity
10     Int: reservedStock ←
11     mm.reservedStock
12     Int: netStock ← savedStock -
13     reservedStock
14
15     // Check if the available net stock
16     is insufficient
17     If: quantityNeeded > netStock
18         // Update reserved stock and
19         create a PIR for the shortage
20         reservedStock ← reservedStock
21         + netStock
22         Int: pirQuantity ←
23         quantityNeeded - netStock
24         createPIR(orderDetail,
25         pirQuantity)
26     Else:
27         // Sufficient net stock, update
28         reserved stock
29         reservedStock ← quantityNeeded
30
31     // Update MasterMaterial with the
32     new reserved stock
33     mm.reservedStock ← reservedStock
34     masterMaterialRepository.save(mm)

```

Algorithm 12. Algorithm of Reserve Stock for Stock-Based Management with Clone-and-Own

Branching and parallel development. The branching and parallel development processes in SPLE tend to have minimal conflicts due to the reference provided by the feature model. In clone-and-own, the branching process can be prone to conflicts without a development reference.

Framework support and documentation. There are certain support features not provided by the framework used in the SPLE method, such as the absence of support for implementing foreign keys and non-primitive data types in ABS microservices.

4. Conclusions

The study underscores the superiority of the SPLE approach over the clone-and-own method, as demonstrated through various comparisons. Utilizing SPLE, software development was conducted efficiently, employing a delta-oriented programming paradigm to create a tailored supply chain management system for manufacturing companies, optimizing production processes. The development process involved integrating previous iterations, adding features, making adjustments, and addressing foreign key implementation. Notably, SPLE enabled streamlined development steps such as the Design Phase, Module Integration, Implementation of Alternative Development Solutions, and Application Variation Testing. Moreover, SPLE exhibited advantages over the clone-and-own approach across several aspects of main issues that consist of lines of code, bug fixing and updating, and support and framework documentation. SPLE required increasing 1-2 lines of code for each product variation, while clone-and-own resulted in multiple duplications of lines of code proportional to the number of features. Bug fixing and updating were more efficient with SPLE, as it only necessitated changing one code base and rebuilding the product to apply changes, whereas clone-and-own demanded altering the entire code base for each variation.

Despite SPLE's superior efficiency in development, it was noted that the ABS microservices framework lacked adequate support compared to Spring Boot, which enjoyed broader support and framework documentation. ABS Microservices exhibited several shortcomings, particularly in the development of foreign keys, queries, non-primitive data types, and inadequate documentation. The author suggests using the winVMJ framework, developed by the Reliable Software Engineering (RSE) team at Fasilkom UI, for future research. The winVMJ framework provides more comprehensive support compared to ABS Microservices. Notable features include support for storing foreign key attributes from other objects and handling null values when no objects are found during queries. By utilizing a more advanced framework, it is expected that the limitations in developing system

variations using the SPLE method can be addressed more efficiently, enabling a more method-focused comparison. Aside from the aforementioned issue from the previous statement, this study also only covers developing *backend* SCM using the SPLE method. The author suggests applying the development *frontend* SCM using the SPLE method.

References

- [1] K. A. Mukhamedjanova, "Concept of supply chain management," *Journal of Critical Reviews*, vol. 7, no. 2, 2020, doi: 10.31838/jcr.07.02.139.
- [2] M. Daneshvar Kakhki and V. B. Gargeya, "Role of information technology in supply chain management: A review," *Journal of Enterprise Information Management*, vol. 32, no. 1, pp. 135–155, 2019.
- [3] T. Khan, M. Mehedi, and H. Emon, "Supply chain performance in the age of Industry 4.0: evidence from manufacturing sector Supply chain performance in the age of Industry 4.0: evidence from manufacturing sector Supply chain performance in the age of Industry 4.0: evidence from manufacturing sector," 2025. [Online]. Available: <https://doi.org/10.14488BJOPM.2434.2025>
- [4] A. Arrumaisha, "The impact of make-to-stock and make-to-order strategy on supply chain performance," *International Journal of Supply Chain Management*, vol. 9, no. 4, pp. 1036–1042, 2020.
- [5] N. R. Sanders, *Supply Chain Management: A Global Perspective*, 5th ed. Upper Saddle River, NJ: Pearson, 2025.
- [6] K. Pohl, G. Böckle, and F. Van Der Linden, *Software product line engineering: Foundations, principles, and techniques*. 2005. doi: 10.1007/3-540-28901-1.
- [7] N. Kingwijati, "Reusable feature identification in software product line engineering," *Journal of Software Engineering and Applications*, vol. 13, no. 6, pp. 292–303, 2020.
- [8] I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella, "Delta-oriented programming of software product lines," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010. doi: 10.1007/978-3-642-15579-6_6.
- [9] R. Hähnle, "The abstract behavioral specification language: A tutorial introduction," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2013. doi: 10.1007/978-3-642-40615-7_1.
- [10] E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen, "ABS: A core language for abstract behavioral specification," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011. doi: 10.1007/978-3-642-25271-6_8.
- [11] S. Apel, D. Batory, C. Kastner, G. Saake, and I. Schaefer, "Feature-oriented software product lines: concepts and implementation," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, pp. 1–45, 2013.
- [12] P. Clements, "A survey of software product line practice," *Software Engineering Institute, Carnegie Mellon University*, 1999.
- [13] J. R. Arnold, S. N. Chapman, L. Clive, and M. Clive, "Introduction to materials management," *Prentice-Hall*, 1991.
- [14] O. Komarudin, H. Arrumaisha, and A. Azurat, "Design and Realisation of Reusable Artefacts for Internal Supply Chain Management in Manufacturing Company," in *Journal of Physics: Conference Series*, 2021. doi: 10.1088/1742-6596/1811/1/012091.
- [15] O. Komarudin, D. Adianto, and A. Azurat, "Modeling requirements of multiple single products to feature model," in *Procedia Computer Science*, 2019. doi: 10.1016/j.procs.2019.11.105.

- [16] M. R. A. Setyautami, R. Hähnle, R. Muschevici, and A. Azurat, "A UML profile for delta-oriented programming to support software product line engineering," in *ACM International Conference Proceeding Series*, 2016. doi: 10.1145/2934466.2934479.
- [17] M. R. A. Setyautami, H. S. Fadhlillah, D. Adianto, I. Affan, and A. Azurat, "Re-Engineering Microservice Applications into Delta-Oriented Software Product Lines," in *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines*, Cham: Springer International Publishing, 2023, pp. 275–292. doi: 10.1007/978-3-031-11686-5_11.
- [18] M. A. Naili, M. R. A. Setyautami, R. Muschevici, and A. Azurat, "A framework for modelling variable microservices as software product lines," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018. doi: 10.1007/978-3-319-74781-1_18.
- [19] A. Novrialdi, D. Adianto, A. Rosyida, P. L. Bagaskara, and A. Azurat, "Towards Erlang-based ABS Microservices Framework for Software Product Line Development," *Jurnal Ilmu Komputer dan Informasi*, vol. 15, no. 2, 2022, doi: 10.21609/jiki.v15i2.1065.
- [20] I. Affan, A. Novrialdi, O. Komarudin, H. S. Fadhlillah, and A. Azurat, "Towards Integrated Access Control for Software Product Line Engineering Web Development," in *Lecture Notes in Networks and Systems*, 2022. doi: 10.1007/978-3-030-82616-1_4.