ICIC Express Letters
Volume **18**, Number **5**, May **2024**

ICIC International ©2024 ISSN 1881-803X
pp. 521–530

SOFTWARE PRODUCT LINE DOMAIN ENGINEERING FOR SUPPLY CHAIN OF SMALL AND MEDIUM ENTERPRISE MANUFACTURER

Oman Komarudin, Eko Kuswardono Budiardjo* and Ade Azurat

Faculty of Computer Science
Universitas Indonesia
Jl. Margonda Raya, Pondok Cina, Depok 16424, Indonesia
oman.komarudin@ui.ac.id; ade@cs.ui.ac.id
*Corresponding author: eko@cs.ui.ac.id

Received May 2023; accepted August 2023

Abstract. *The Industry 4.0 era is challenging manufacturing industries to improve their flexibility and responsiveness to change various needs by upgrading their operations strategies, methods, and technologies. Software product line engineering (SPLE) is a promising software development paradigm that satisfies those needs. Several industries that have adopted this concept have had promising results. However, different industries, especially small and medium enterprise (SME) manufacturing, have applied different approaches because there is no standardized flow applicable for different types of settings. This study arranges procedures for implementing SPLE in certain domains with an extractive approach. Requirements engineering is carried out at each SME manufacturer referring to their respective supply chains. Each identified application feature is mapped to build a feature model. The extended unified modeling language for delta oriented programming profile is used to represent the results of requirement gathering for a more abstract architectural design. The result of this research is an approach to performing domain engineering of SPLE for SME manufacturing.*

Keywords: Software product line engineering, Software engineering, Internal supply chain, Small and medium enterprises, Manufacturing

1. Introduction.

Industry 4.0 is challenging industries to keep up with changes continuously. Changes can be made by improving strategy, methods, or technology [1]. Complex and continuously changing business environments and customer demands, as well as the need for flexibility and responsiveness in companies, are training the spotlight on to advanced technological improvements to supply chains [2]. Forecasting tools and supply chain data can help predictive maintenance be planned in advance, reducing downtime. Supply chain software enables manufactures to support their activities and enhance the productivity and profits [3]. Manufacturers are encouraged to endorse their strategies and software solutions to attain competitive performance goals [4].

Enterprises resources planning (ERP) and other commercial off-the-self (COTS) proprietaries provide software solutions for supply chain requirements [5]. Manufacturers use ERP systems as software solutions to effectively manage their business processes. The effective adoption and deployment of ERP systems is essential for the survival and competitiveness of enterprises [6]. Alaskari et al. [6] propose a framework for implementation of ERP in small and medium enterprises (SMEs). However, it is extremely difficult to choose and operate an ERP system, particularly given the large costs involved in the procedure. Furthermore, SMEs typically make small investments in technology [7]. Tailored software development is a common industry practice for SMEs to meet the needs of supporting software and considering the simplicity, maintainability and certification of the software [8].

DOI: 10.24507/icicel.18.05.521

Manufacturing companies are similar in their supply chain activities, producing goods from raw materials into final products. Generally, their activities consist of raw goods procurement, production processes, and distribution [9]. The similarities of manufacturing processes provide a great possibility in the creation of software through clone-and-own. Developers create software variants with similar needs by adopting existing software and then customizing it to meet new requirements. The result of this clone-and-own activity is several software products derived from the same code, but the end result is several software variants that can be very different from other products [10].

Corporate strategy changes over time, following continuous developments caused by both internal and external changes. This development aims to maintain the continuity of the company [11]. The requirements constantly increase in complexity and diversity due to more demanding customers, evolving technologies, and business strategies [12]. As with SMEs in general, SME manufacturers supply chains operate in dynamic and rapidly changing environments to maintain their existence [6]. These changes in strategy drive modifications to the software utilized by each SME manufacturer. Software created through a clone-and-own technique might be increasingly distinctive from the original. The more variants that are made, the more complex the documentation and maintenance [8].

Software product lines aim to support the development of variant software through systematic reuse of shared assets [13, 14, 15]. SPLE by its nature is a software development paradigm that allows exploitation of various software requirements' commonalities and variabilities, thus making them into reusable assets [16]. This reusability feature contrasts with single system development and gives SPLE more advantages in flexibility for future development when changes are inevitable [10]. SPLE is an efficient method of developing a range of related products. It maximizes product commonalities while still acknowledging their differences, unlike traditional product-centric development [14].

This research focuses on how to implement SPLE in the supply chain of SME manufacturers. Based on Porter's value chain [9, 11, 17], the internal supply chain is the extensive factor influencing the company's business strategy. The internal supply chain consists of processes: procurement of raw material, production processes, and distribution. This research outlines the steps for applying SPLE to internal supply chain processes of SME manufacturing.

The following organizational structure is used in this essay. Section 1 outlines cutting-edge literature on software product line engineering for SMEs. Section 2 presents the fundamental literature on software product line processes. Section 3 discusses how to use standard tools to perform SPLE for SME manufacturers. This section describes how to use UML-DOP (unified modeling language-delta oriented programming) to determine requirements in an SPLE manner and build software architecture. Section 4 depicts the evaluation of the suggested framework. Finally, Section 5 concludes the research.

2. **Software Product Line Engineering.**

SPLE has gained significant attention over the years as a strategic approach that incorporates business, organization, and technology in such a way as to achieve high effectiveness and efficiency [19, 20]. SPLE categorizes its development phase into two major domains [15, 16]: domain engineering, and application engineering. Domain engineering is responsible for creating a reusable platform from various requirements, whereas application engineering obtains applications from a reusable platform based on the needs of each software product line.

Variability models are created in the early stages of domain engineering based on requirements. The domain implementation translates the variability of the model into architectural design. Domain realization implements the reusable software components of the architectural design. Requirements analysis explores the demands of individual customers as part of application engineering. Product derivation is the production process of application engineering in which reusable artifacts are combined based on the results of the requirements analysis.

Domain analysis is a form of requirements engineering that applies to an entire product line. This step determines the domain's scope, that is, which products should be covered by the product line. Domain analysis also determines what features are relevant and should be developed as reusable artifacts. An extractive approach is the most common way to adopt a software product line with many system variants [21].

To determine the scope for SMEs SPLE, advanced user-based collaborative filtering (advanced UCF) is proposed. The advanced UCF algorithm is an enhancement to the UCF algorithm that takes SMEs' characteristics into account while scoping the needs for SPLE. Advanced UCF is based on the UCF and is thought to be more appropriate for application in this SME case study [22]. In line with advanced UCF, Komarudin et al. [24] define the domain analysis steps as follows: 1) recognize all features of all requirements from each business process; 2) categorize them into feature groups depending on their areas of duty; 3) map each feature's presence across all products on a product roadmap matrix; 4) analyze the resulting product roadmap to find commonality and variability features; 5) transform the results into a feature model.

The results of domain analysis are typically described in a feature model [16]. With the introduction of the cardinality idea to a feature model, the cardinality based feature model (CBFM) as an extension of the original feature oriented domain analysis (FODA), is more expressive in describing commonality and variability in SPLE [25].

In every software development project, defining the requirements and designing software architecture are inextricably linked processes [26]. The variability modeling that is defined on the domain requirements engineering is used as an input reference to determine the software architecture [15]. The translation between variability modeling and the software architecture development process is a challenge in SPLE [20]. The architectural design must be able to accommodate the flexibility that the SPLE concept uses to its advantage [27]. The design must also accommodate changes and additions to requirements in the future [28].

Several approaches in mapping variability models into implementation artefacts have been proposed. The conditional compilation (CC) concept uses #ifdefs to represent the variability model. Feature-oriented programming (FOP) proposes a more systematic approach to creating reusable artefacts than CC, which uses #ifdefs in the code [29]. FOP maps the features in the variability model using feature modules [16]. Delta oriented programming (DOP) implements variability models of SPL using the delta approach [30]. DOP represents the product line using a core module and sets of delta modules that allow a flexible "n-to-m" mapping of feature variants in variability model.

Unified modeling language (UML) is a standard approach for modeling a software system. Object management groups (OMGs) provide a mechanism to allow modification of UML syntax in particular application domains, called a UML profile [8]. A UML profile is a set of extensions that allow customization: stereotypes, tagged values, and constraints. Setyautami et al. [31] proposed a mechanism utilizing the UML profile to represent the architecture of DOP that supports SPL, called a UML-DOP profile. Each element in DOP extends a UML meta class and is translated to stereotypes, tagged values, and constraints.

3. **Software Product Line Engineering for Supply Chain of SME Manufacturers.** This section describes the implementation of domain engineering for internal supply chains SME manufacturers based on Figure 1. The initial phase consists of conducting domain requirements engineering to identify the feature model, which is subsequently translated into architectural design and realization.

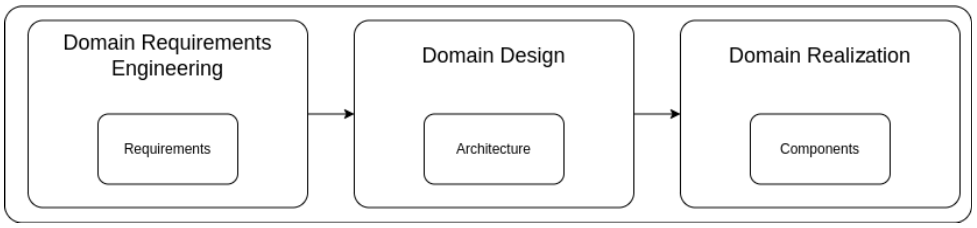


FIGURE 1. Domain engineering process for supply chain of SME size manufacture [15]

3.1. **Domain requirements.** This subsection describes the requirement elicitation process for each company with its own characteristics. This process analyzes the requirements of the end-to-end business flow that occurs within a manufacturing company. We capture and analyze the similarities and differences of each company to construct a requirements artifact.

3.1.1. *Identification of the business requirements.* The activities of a manufacturing company consist of receiving and processing product demands, managing inventory, processing materials into products, and delivering products to customers [23]. Even though manufacturers engage in common activities, each manufacturer has a different way of carrying out its activities, affecting the needs of the company in the system. The difference in the needs of each company lies in how each department performs its activities. This difference is due to many factors such as business strategy, market conditions, internal policies, and various external factors [24].

Based on the general needs of SME sized manufacturing internal supply chains, we categorize these needs into three main business processes: 1) material management: this process handles the materials used in manufacturing, including raw materials, semi-finished goods, finished goods, and other material components; 2) planning and control: this process involves planning production activities and managing inventory to meet company needs, includes managing material requirements, processing materials as per demands, and receiving materials in the warehouse; 3) production: the core activity of manufacturing, where raw materials are processed to create finished products. This process also includes quality control.

3.1.2. *Feature identification.* We analyze three different manufacturing company requirements based on current activities and supporting systems. The manufacturers involved in this research are referred to KI, OI, and KF. Each represents a different type of production: KI manufactures custom metal molds, OI manufactures consumer electronics, and KF produces food. We explore the needs of the three companies and then extract the needs of each feature.

To demonstrate the proposed method, we focus on three manufacturers with different demand management requirements. KI specializes in making molds based on customer orders. Their planning and production happen when they receive an order, and they do not keep finished goods in inventory, only raw materials. Quality inspection occurs once the finished product is ready. KI uses a single structured bill of materials and does not reuse materials. Rejected products are reworked into finished goods or disposed of as scrap.

OI processes plastic materials into electronics sub-parts and assembles them with other parts to make finished products. They have a continuous production process to meet market demand, with a production plan that considers finished goods inventory and received orders. Quality inspection is conducted in each sub-process before sending the product to the next stage. OI uses a multilevel bill of materials and can reprocess rejected plastic products as additional raw materials.

KF is a food manufacturer that produces foods from raw materials. Their production plan is based on orders and stock estimation to prevent overstocking finished goods. KF uses a multilevel bill of materials to separate food production, packaging, and packaging processes. Quality inspection is performed in each sub-process to avoid sending rejected food to the packing stage. Rejected products cannot be reworked or used as additives; they are scrapped and destroyed in KF.

Table 1. Feature identification

Root feature requirement	Feature	Variant
Material Management	BOM Structure	Simple
		Hierarchical
	Material Usage	Pure Material
		Reuse Material
Planning and Control	Demand Management	Make-to-Order
		Make-to-Stock
	Inventory	Stock-based
		Planned
Production	Production Process	Single Process
		Complex Process
	Quality Inspection	Each-process
		End-of-process
	Rejection	Scrap
		Rework

3.1.3. *Feature and software product matrix.* Table 1 shows the complete features of an internal supply chain. We map the features onto software products based on the manufacturer's requirements. The matrix maps *root features* and *features* onto the software product. We use a check mark (✓) to show the feature that should be available in a software product. Table 2 maps the presented feature of every software product.

Table 2. Matrix of features and software products

Feature	Variant	Company KI	Company OI	Company KF
BOM Structure	Simple	✓	–	–
	Hierarchical	–	✓	✓
Material Usage	Pure	✓	✓	✓
	Reuse Material	–	✓	–
Demand Management	Make-to-Order	✓	✓	✓
	Make-to-Stock	–	✓	✓
Inventory	Stock-based	✓	✓	✓
	Planned	–	✓	✓
Production Process	Single Process	✓	✓	✓
	Complex Process	–	✓	✓
Quality Inspection	Each-Process	–	✓	✓
	End-Process	✓	✓	✓
Rejection	Scrap	✓	–	✓
	Rework	–	✓	–

3.1.4. *Feature modeling.* We implement the feature model based on the identified software product and features. This process consists of two activities: identifying the commonalities and variability of software products, and creating a feature model using a feature diagram.

Once we identify the features and map them to the software product, we can determine their commonality and variability. Features found in all categories are considered common and are mandatory for all software products. Some features are specific to certain categories and are called varying features. These varying features are optional and can be chosen or omitted. Additionally, some features may rely on other features to work properly.

The features used in all software products are *Make-to-Order* in the *Demand Management* feature, *Pure* in the *Material Usage* feature, *Stock-Based* in the *Inventory* feature, and *End-Process Quality Inspection*. A feature that exists but is not used by some software product is not mandatory.

Based on the description, we can model the features using a feature diagram as seen in Figure 2. The feature diagram models the commonality and variability of the software product line for manufacturers. It contains mandatory and optional features that can be found in a product. It also describes the constraints and the requirements of a feature in the software product line.

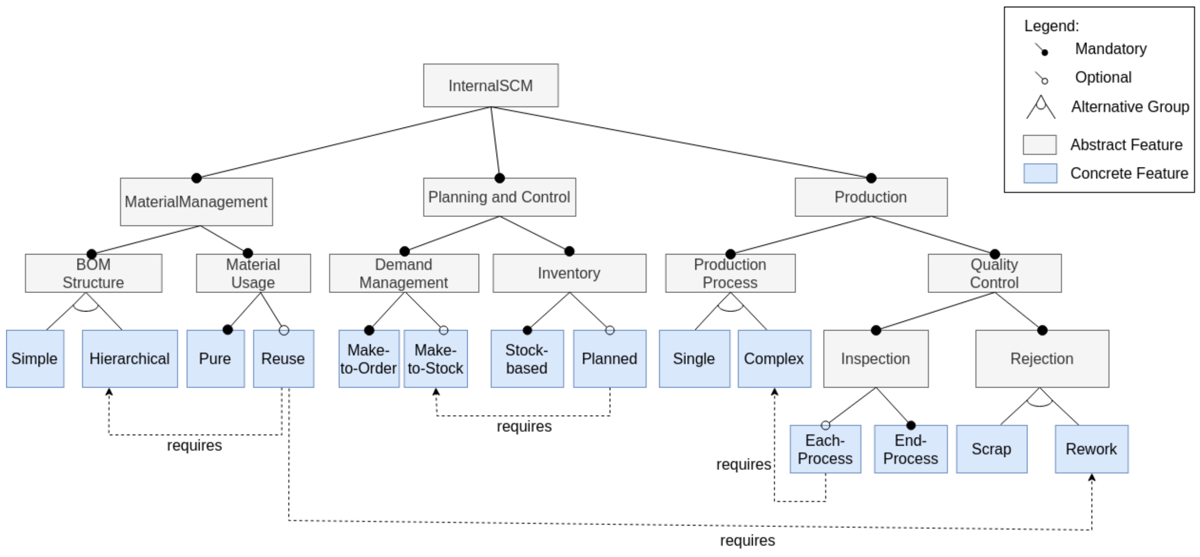


FIGURE 2. Feature model of SPL for manufacture internal supply chain

Figure 2 depicts the feature model of a manufacturing internal supply chain. The feature diagram clearly illustrates the model of the software product line. This model shows the abstract and concrete features needed in the software product for manufacturers. The commonality and variability shown in the diagram and dependencies are indicated by the `required` relationship.

3.2. **Domain architecture.**

Product line architecture is produced in the domain design stage [15]. The variability modeling that is defined by the domain requirements engineering can be used as an input to determine the reference architecture. In this study, we use a UML-DOP profile to model variations in the UML notation [31]. We propose several levels of abstraction at the architectural model.

3.2.1. *Component diagram.* First, we use a component diagram to capture the variations at the feature level. In the UML-DOP, a feature is modeled as a UML component with stereotype `<<feature>>`, so we transform the feature diagram into a UML component

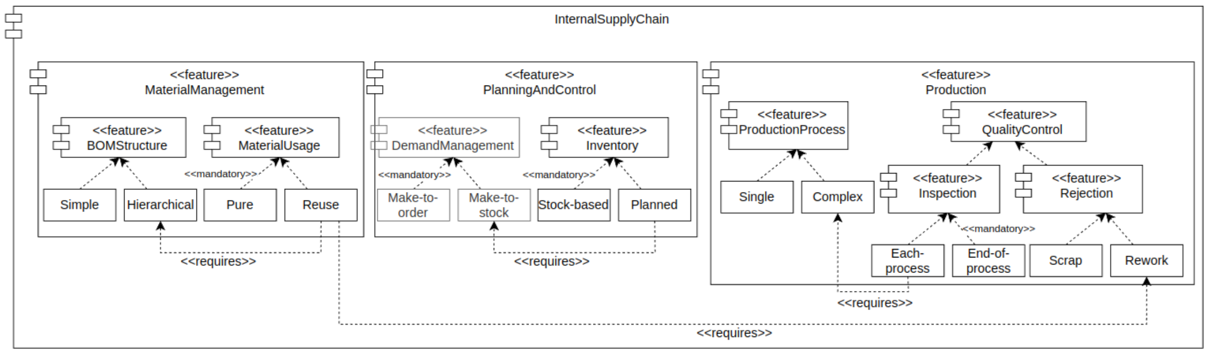


FIGURE 3. Component diagram: Feature model for internal supply chain of SME size manufacture

diagram. Figure 3 depicts the feature model on a UML-DOP profile. The stereotype `mandatory` also represents the mandatory feature of the feature model. The stereotype `requires` represents the dependency between features.

Starting from this point, we will focus on the demand management feature to demonstrate how these methods work. Demand management has one mandatory feature and one optional feature. In addition, the demand management feature interacts with other features in inventory and checks orders. It can represent the process of design and realization for other features.

The *Make-to-Order* strategy adds the customer orders into the material planning, while the *Make-to-Stock* strategy combines the order and inventory to create material planning. Demand management processes the requirements from sales orders, so it is connected to the `Sales and Distribution` component. Furthermore, `DemandManagement` is also connected to the `Warehouse` feature.

Figure 4(A) shows part of a UML component diagram for a company that uses a *Make-to-Order* approach in demand management. The `MDemandManagement` module has an association with the `MOrderDetail` module. A variation occurs when a company uses a *Make-to-Stock* approach (Figure 4(B)). Furthermore, the `MDemandManagement` module not only has an association with the `MOrderDetail` module, but also with the `MInventory` module. Figure 4(C) transforms both into an SPL component diagram for `DemandManagement`.

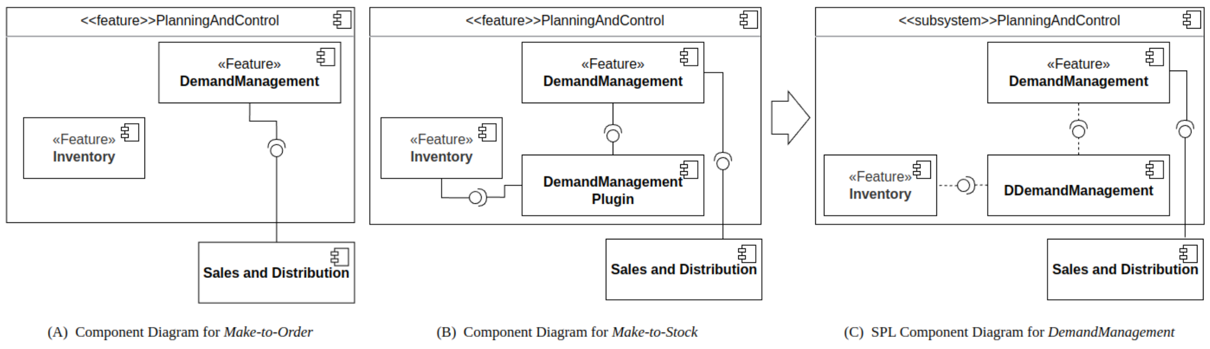
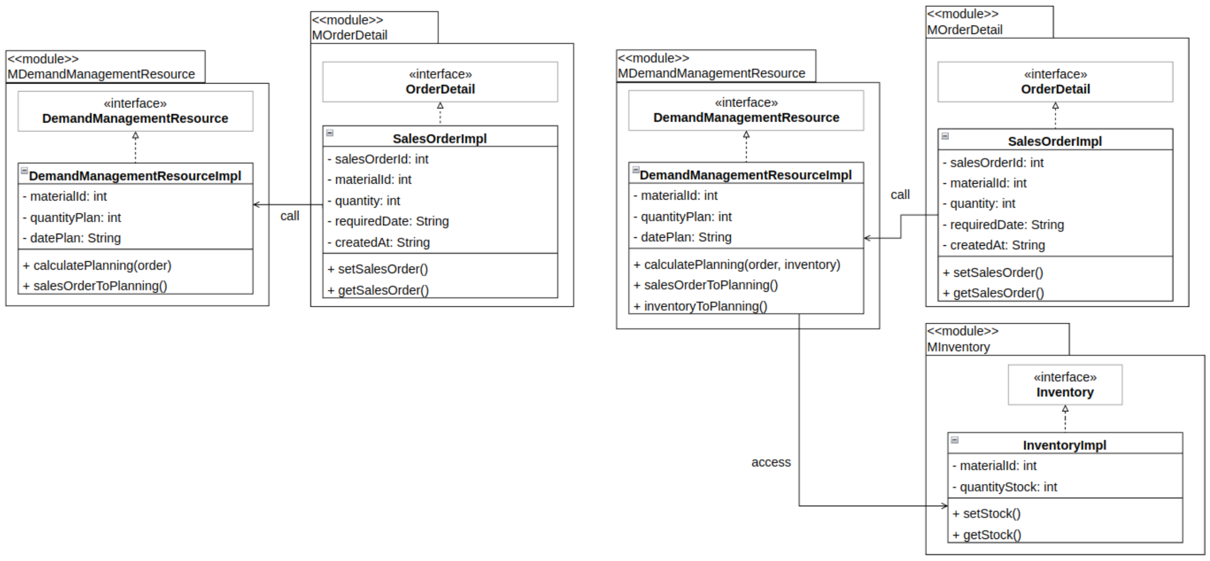

<<feature>>PlanningAndControl
«Feature» DemandManagement
«Feature» Inventory
Sales and Distribution
(A) Component Diagram for Make-to-Order
<<feature>>PlanningAndControl
«Feature» DemandManagement
«Feature» Inventory
DemandManagement Plugin
Sales and Distribution
(B) Component Diagram for Make-to-Stock
<<subsystem>>PlanningAndControl
«Feature» DemandManagement
«Feature» Inventory
DDemandManagement
Sales and Distribution
(C) SPL Component Diagram for DemandManagement

FIGURE 4. Component diagram of DemandManagement

3.2.2. *Class diagram.* In the architectural stage, we analyze the variations and then extract those two UML class diagrams into the UML class diagram with the UML-DOP profile. Figure 5 shows the variations of *DemandManagement* class diagram. We place the common attributes and operations in the core module and the variants in the delta module. As defined in the DOP paradigm, the delta module can change the implementation by adding, removing, or modifying elements.



(A) Class Diagram for *Make-to-Order* (B) Class Diagram for *Make-to-Stock*

FIGURE 5. UML class diagram for each DemandManagement strategy

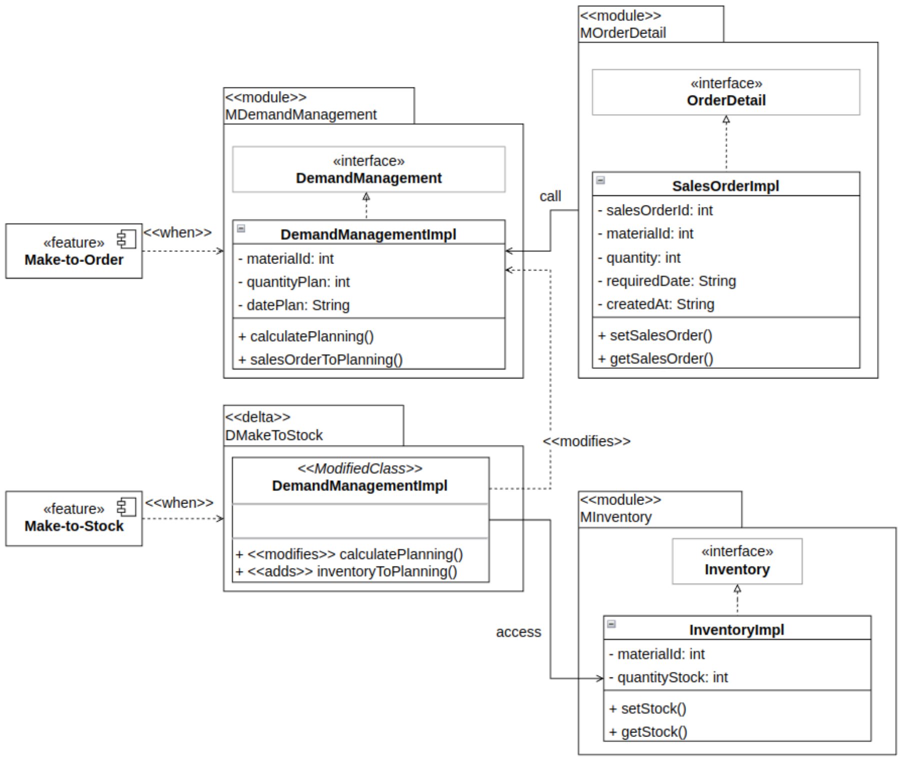


FIGURE 6. Class diagram for demand management

Figure 6 shows the UML class diagram for `DemandManagement`. The core module is represented by the `MDemandManagement` module. This module contains the `DemandManagementImpl` class, which implements the `DemandManagement` interface. The class consists of common attributes and operations in the `DemandManagement` module, such as `calculatePlanning()`.

Variants of this class are placed in the delta module. As shown in Figure 6, the package `DMakeToStock` represents a delta module. This delta module changes the `DemandManagementImpl` class by modifying one method and adding a new method. This is denoted by a class with the stereotype *modifiedClass*.

4. **Evaluation.**

In domain requirement engineering, we follow several steps, including business process analysis, feature identification, software product categorization, mapping features to software products to find similarities and differences, and creating a variability model. Each step in this process is interconnected and essential.

For domain design, we use the UML-DOP profile to build the SPLE architecture, specifically designed for SME manufacturers. The architecture is based on the feature model established during domain engineering. We use component diagrams to show the structure of the variability model and how features are connected. This approach improves modularity and makes software development and maintenance easier. By using abstraction, we isolate specific functions within separate components, making them distinct and easy to interact with. These components can be implemented, developed, replaced, or reused independently. Transforming mandatory and optional features into separate components connected through interfaces allows for flexible architectural-level changes.

5. **Conclusion and Future Work.**

This work shows that software development using SPLE is possible. The study extracts information from three manufacturers with their internal supply chains. By analyzing their commonality and variability, representative model features are created. The UML-DOP profile is crucial in the engineering design process to build the domain architecture.

The domain architecture uses UML component diagrams with stereotypes to present the feature model in a more abstract form. Class diagrams with a UML-DOP profile make it easier to describe how features work and connect to each other. Using delta oriented programming allows changes to the core module of one feature to generate feature variants.

Future work could involve implementing application engineering for the same domain, and testing the created code could be another avenue of research.

Acknowledgment. This work is fully supported by research of Reliable Software Engineering Lab of Universitas Indonesia. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] R. Y. Zhong, X. Xu, E. Klotz and S. T. Newman, Intelligent manufacturing in the context of Industry 4.0: A review, *Engineering*, vol.3, no.5, pp.616-630, DOI: 10.1016/J.ENG.2017.05.015, 2017.

[2] H. Fatorachian and H. Kazemi, Impact of Industry 4.0 on supply chain performance, *Production Planning and Control*, vol.32, no.1, pp.63-81, DOI: 10.1080/09537287.2020.1712487, 2021.

[3] E. A. Maria, K. A. Maria and M. A. Alia, Smart agents in the business information system, *ICIC Express Letters*, vol.13, no.10, pp.921-929, DOI: 10.24507/icicel.13.10.921, 2019.

[4] K. Li and J. Gao, A framework to integrate manufacturing information systems, *Engineering*, nos.11-12, pp.22-23, DOI: 10.1007/978-0-387-49864-5, 2014.

[5] V. K. Agrawal, V. K. Agrawal and A. R. Taylor, Trends in commercial-off-the-shelf vs. proprietary applications, *Journal of International Technology & Information Management*, vol.25, no.4, pp.1-35, 2016.

[6] O. Alaskari, R. Pinedo-Cuenca and M. M. Ahmad, Framework for implementation of enterprise resource planning (ERP) systems in small and medium enterprises (SMEs): A case study, *Procedia Manufacturing*, vol.55, no.C, pp.424-430, DOI: 10.1016/j.promfg.2021.10.058, 2021.

[7] T. Oliveira, M. Thomas and M. Espadanal, Assessing the determinants of cloud computing adoption: An analysis of the manufacturing and services sectors, *Information and Management*, vol.51, no.5, pp.497-510, DOI: 10.1016/j.im.2014.03.006, 2014.

[8] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon and A. Egyed, The ECCO tool: Extraction and composition for clone-and-own, *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol.2, pp.665-668, DOI: 10.1109/ICSE.2015.218, 2015.

[9] A. Ivanov and T. Jaff, Manufacturing lead time reduction and its effect on internal supply chain, *International Conference on Sustainable Design and Manufacturing*, vol.52, DOI: 10.1007/978-3-319-32098-4, 2017.

[10] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker and K. Czarnecki, An exploratory study of cloning in industrial software product lines, *Proc. of the European Conference on Software Maintenance and Reengineering (CSMR)*, pp.25-34, DOI: 10.1109/CSMR.2013.13, 2013.

[11] H. Paul, *Business Process Change, Third Edition: A Business Process Management Guide for Managers and Process Professionals*, Morgan Kaufmann Publishers Inc., 2014.

[12] F. Stallinger, R. Neumann, R. Schossleitner and S. Kriener, Migrating towards evolving software product lines: Challenges of an SME in a core customer-driven industrial systems engineering context, *Proc. of International Conference on Software Engineering*, pp.20-24, DOI: 10.1145/1985484.1985490, 2011.

[13] L. Northrop and P. Clements, *A Framework for Software Product Line Practice, Version 5.0*, Software Engineering Institute, 2012.

[14] P. C. Clements, Product line engineering comes to the industrial mainstream, *INCOSE Int. Symp.*, vol.25, no.1, pp.1305-1319, DOI: 10.1002/j.2334-5837.2015.00131.x, 2015.

[15] K. Pohl, G. Böckle and F. Van Der Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer Berlin Heidelberg, 2005.

[16] S. Apel, D. Batory, C. Kästner and G. Saake, *Feature-Oriented Software Product Lines*, Springer, 2013.

[17] C. Basnet, The measurement of internal supply chain integration, *Management Research Review*, vol.36, no.2, pp.153-172, DOI: 10.1108/01409171311292252, 2013.

[18] D. Postmus, *The Supply Chain of Enterprise Software: Strategy, Structure, and Coordination*, Ph.D. Thesis, University of Groningen, Groningen, 2009.

[19] D. Amalfitano, V. De Simone, A. R. Fasolino, M. Lubrano and S. Scala, Introducing software product lines in model-based design processes: An industrial experience, *Proc. of 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016)*, pp.287-290, DOI: 10.1109/WICSA.2016.36, 2016.

[20] A. Metzger and K. Pohl, Software product line engineering and variability management: Achievements and challenges, *Futur. Softw. Eng.*, pp.70-84, DOI: 10.1145/2593882.2593888, 2014.

[21] C. W. Krueger, Easing the transition to software mass customization, *The 4th International Workshop on Product Family Engineering (PFE-4)*, vol.2290, no.512, pp.282-293, DOI: 10.1007/3-540-47833-7_25, 2002.

[22] N. M. S. Iswari, E. K. Budiardjo and Z. A. Hasibuan, E-business applications recommendation for SMES using advanced user-based collaboration filtering, *ICIC Express Letters*, vol.15, no.5, pp.517-526, DOI: 10.24507/icicel.15.05.517, 2021.

[23] M. Nakano and K. Matsuyama, Internal supply chain structure design: A multiple case study of Japanese manufacturers, *International Journal of Logistics Research and Applications*, vol.24, no.1, pp.79-101, DOI: 10.1080/13675567.2020.1726305, 2021.

[24] O. Komarudin, D. Adianto and A. Azurat, Modeling requirements of multiple single products to feature model, *Procedia Comput. Sci.*, vol.161, pp.107-114, DOI: 10.1016/j.procs.2019.11.105, 2019.

[25] Wahyudianto, E. K. Budiardjo and E. M. Zamzami, Feature modeling and variability modeling syntactic notation comparison and mapping, *Journal of Computer and Communications*, vol.2, no.2, pp.101-108, DOI: 10.4236/jcc.2014.22018, 2014.

[26] B. Unhelkar, *Software Engineering with UML*, Taylor & Francis Group, 2018.

[27] J. Martinez, T. Ziadi, J. Klein and Y. Le Traon, Identifying and visualising commonality and variability in model variants, in *Modelling Foundations and Applications. ECMFA 2014. Lecture Notes in Computer Science*, J. Cabot and J. Rubin (eds.), Cham, Springer, DOI: 10.1007/978-3-319-09195-2_8, 2014.

[28] V. Rajlich, Software evolution and maintenance, *Future of Software Engineering (FOSE'14)*, pp.133-144, DOI: 10.1145/2593882.2593893, 2014.

[29] G. Coutinho, S. Ferreira, F. Nunes, E. Figueiredo and M. D. Almeida, On the use of feature-oriented programming for evolving software product lines – A comparative study, *Science of Computer Programming*, vol.93, pp.65-85, DOI: 10.1016/j.scico.2013.10.010, 2014.

[30] I. Schaefer, L. Bettini, V. Bono, F. Damiani and N. Tanzarella, Delta-oriented programming of software product lines, in *Software Product Lines: Going Beyond. SPLC 2010. Lecture Notes in Computer Science*, J. Bosch and J. Lee (eds.), Berlin, Heidelberg, Springer, DOI: 10.1007/978-3-642-15579-6_6, 2010.

[31] M. R. A. Setyautami, R. Hähnle, R. Muschevici and A. Azurat, A UML profile for delta-oriented programming to support software product line engineering, *Proc. of the 20th Int. Syst. Softw. Prod. Line Conf.*, pp.45-49, DOI: 10.1145/2934466.2934479, 2016.