Design and Realisation of Reusable Artefacts for Internal Supply Chain Management in Manufacturing Company

Oman Komarudin\textsuperscript{1\*}, Hanifa Arrumaisha\textsuperscript{2}, and Ade Azurat\textsuperscript{3}

\textsuperscript{1,2,3}Faculty of Computer Science, Universitas Indonesia, Depok, Indonesia

\*oman.komarudin@ui.ac.id

Abstract. Maintaining many similar software products with huge similarities and some variations have been studied is Software Product Line Engineering (SPLE). Traditional software methodology lacks the notions to model and maintain those variability and commonality in all phases of standard known software methodology (e.g. from requirements phase up to the testing phase). This paper shows a proposed software methodology based on SPLE to ease the development and maintainability of similar software product. The paper focuses on the design and realizes the reusable artefacts to generate a family of software products systematically with case study example of Supply Chain Management in Manufacturing. This methodology utilized the commonly known UML diagram with the use of UML Profile of Delta Programming and some tools of code generator.

1. Introduction

Pohl et al. [1] defines SPLE as a software development paradigm that exploits the commonality and variability of various software requirements. SPLE Process consists of domain engineering and application engineering process. Domain engineering is responsible to create reusable platform from various requirements, while application engineering derive applications from reusable platforms based on the needs of each software product line. Variability models are made in the early stages of the domain engineering based on requirements. Domain design will translate model variability into architectural design. Domain realization implements reusable software components from the architectural design.

In a single software development, defining requirements and developing software architecture are very closely related. Requirements are the basis for creating a software architecture design. There is a challenge on how to develop software architecture in SPLE to accommodate the variability and bridge the various requirements with software development [2]. In SPLE, the domain design create the software architecture based on the variability model.

The architectural design must be able to accommodate flexibility as an advantage of the SPLE concept. The design must also be considered to accommodate changes and additions to the requirements of SPLE products in the future. Delta Oriented Programming (DOP) supports SPLE paradigm by implementing delta and core module concept [3]. The UML-DOP Profile enrich the current UML notation to support the delta-oriented programming [4].

This work extend our previous work about modeling requirement of multiple single product to feature model [5] The previous work results a method to create feature model from various single product. This work proposes a methodology to design and realise the reusable artefacts as part of the
domain engineering process in SPLE. The domain design process describes the transformation from feature models into domain design. The domain realisation describes the realisation from architectural design into reusable artefacts. The motivation of this work is to provide a systematic scheme to develop the reusable platform from requirement. Manufacturing demand management will be used to show the running example of this work.

The activities of manufacturer consists of sales and distribution, material planning, production processes, purchasing, and inventory. Sales and distribution capture the market demand and deliver the products to customers. Material planning ensures the fulfillment of market demand at the right time. The production process converts raw materials into products according to customer requirements. Purchasing ensures raw materials inventory can always meet production needs. Inventory control gives the best performance of manufacturers creating the products but avoiding the maintenance costs of unneeded supplies. Even though manufacturers have a common activity, each manufacturer has differences in carrying out their activities. This difference is due to many things such as business strategy, market conditions, internal policies, and other external factors [5]. This situation makes the development of SPLE for manufacturers is very promising.

Currently, we have found some practices by industry in applying SPLE [6]. Some industry has report successful benefit of applying SPLE. However, none of them describes concrete methodology that can be applied by other in different setting.

This work based of UML-DOP on SPLE to abstract class diagrams to provide coherent support between design and reusable implementation. The proposed methodology of Software Product Line Development will be described with the emphasize on the design of reusable artefact to support commonality and variability.

2. Proposed Method
This work proposed a method to realize the development process of reusable artifact. It covers all processes on developing software product line engineering from requirements, design, and realization. This proposed method also considers variability on each process with involvement of OVM and UML-DOP Profile. The Figure 1 depicts the proposed method.

![Figure 1. Research Steps](image)

As shown in Figure 1, each process produces different artefacts. Design and realisation processes produce a leveling architecture, includes package diagram and component diagram with component plugin in design process, and class diagram in realisation process. This leveling architecture enables
abstraction in package level and component level. Thus, architect and developer can focus on what
they want to observed, includes its variabilities related to the architecture level.

This work carried out the steps in several stages to develop software product line engineering:

1. Abstracting the feature into package diagram involves OVM related
   Design of package diagram capture large functionality of system and its interaction. Then involves
   OVM related to show a package is handled those variabilities marked by involvement of OVM.

2. Defining component diagrams of each packages with component plugin
   Design of component diagrams are the detail solution of a package. It used component plugin to
   wrap solutions related to a specific variation point, so the architecture are support for flexibility
   and maintainability on handling variabilities. The design starts with core component first, then
   followed by design of plugin component.

3. Describing the details of development in the form of class diagrams
   Class diagrams are designed to choose the best approach on implementing delta oriented
   programming to solve the problem and its variabilities. Design process use UML-DOP Profile to
   modelling the class diagram with considerations of variabilities, core, and delta module. The
   design also starts with core module first, then followed by delta modules.

4. Implements the class diagrams into core and delta modules
   The last steps is implements the class diagram into codes into core modules first, then followed by
   delta modules.

3. Feature Modeling

Variability is main element for software product line engineering. Feature model describes variability
of the system. It used to traces variability in architecture for this works.

In feature demand management, demand management process calculates all product requirements
into a Planned Independent Requirement (PIR). PIR calculation depends on the type of strategy group
used. When using the make-to-order strategy, requests from consumers will immediately increase the
amount of PIR in accordance with the planning period and material needed. In the make-to-stock
strategy, requests from consumers will be processed by checking the stock in the warehouse first.
Then the insufficient stock will increase the PIR in the same period.

Figure 2 shows a snippet of feature model of demand management using OVM. This feature model
are obtained from requirement engineering process. Demand Management as variation point has two
variants: OrderBased; and StockBased.

![Figure 2. Orthogohnal Variability Model of Demand Management](image)

4. Domain Architecture

This work will show how to interpret the requirements into architectural design. Different from the
architectural design of a single product, this architectural design must be able to depict the
requirements structure of each software product. This architectural design must also have the ability to describe the relationships between components in the software product (texture). The relationship of the components can be a dependency or constraints.

4.1. Package Diagram
Package diagram is high level architecture that captures large functionality of system and its interaction. Then, component diagrams handle the detail of each package. The difference between this package diagram and package diagram of single system is the architecture involves variation points or variants of OVM to show a package is handled variability.

Figure 3 depicts the interrelation between packages in a package diagram. As mentioned in section 1, the activities of manufacturer consist of sales and distribution, material planning, production processes, purchasing, and inventory. Each activity represents requirements of single functionality of system for end users, thus, it is represented as packages.

![Figure 3. Package Diagram of Demand Management Feature](image)

In the case of feature demand management, as shown in figure 3, package diagram describes fulfillment demand strategy variation point is handled in package MaterialPlanning. Therefore the detail solution about the variation point is turned over component diagram, since the variability of fulfillment demand strategy does not affect the package diagram to create a specific package for its variant.

4.2. Component Diagram
Component diagram is a more detail design than package diagram. Design of the component diagram in this work is following two steps, designs the core components and its connection to other components, then designs the plugin components and its connection based on variability point.

Regarding to the requirements, the make-to-order strategy adds the customer orders into the material planning, while make-to-order strategy combines the order and inventory to create the material planning. From the requirement engineering process, OrderBased variant is a core module. Thus, component diagram will designed the core component of feature demand management as OrderBased variant of fulfillment demand strategy variation point.
4.2.1. Core component. The first step is designing the core components and its connection to other components. Feature DemandManagement is created as a core component with feature stereotype that represent a feature of feature model. It has a solid line to show that it must be included in every product family. Demand management processes the requirements from sales to orders. Therefor, an interface that is connected to the Sales and Distribution subsystem needs to be made. The component diagram of demand management shown in figure 4.

![Figure 4. Core Component of Demand Management](image)

4.2.2. Plugin Component. The second step is designing the plugin components and its connection based on variability point. This work designs component diagram using component framework with component plugin concept to handle variability in SPL. Plugin component wraps solutions of a variation point within a core component.

The concepts of core and delta modules in delta oriented programming are used in designing plugin component. Its concepts used as foundation of the concept core and plugin components. Core components are correlated with core modules and plugin components are correlated with delta modules. The OVM involved in the architecture gives the explanation which variant used to construct a core component, and its variation point are configured in the plugin component. As shown in figure 5 that the demand strategy plugin handles fulfillment demand strategy variation point.

![Figure 5. Plugin Component of Demand Management](image)
The plugin connects to its core component using interface and its solid line to the required interface. Interface hides the implementation of the connected component. Therefore, the core component does not need to know which variant or delta module are selected during the product selection process. Then, designs of the plugin is based on the requirements of its variant related to the plugin.

The use of core and plugin component makes the architecture support flexibility. When new variants are detected in the future, it only affect to the realisation of plugin component. Thus, the architecture is flexible to accept new variant without affect other component. Such as, an interface connects the plugin to Warehouse Feature for the StockBased variant.

Core and plugin component also support the architecture with maintainability. When new variation points are introduced in the future, new plugin component will added to the component diagram then define its behaviour. The new variation point that has already defined core component will not affect its core component. Variant implemented by its core component can be detected with attachment of OVM. Therefore, core component may related to many variability point without modifies it. Those design are compatible with core and delta module concept in DOP.

Thus, the component diagram in this work can abstracts core and delta module of the realisation. It also capture delta oriented programming concept in the level of component. Then, architect can view the software product line system at the lower detail level and can decide which component to reuse wisely.

5. Domain Realisation

This work shows how to realize the architectural design into class diagram then implement it using SPLELive Engine. The realization must obey the concepts of delta module and core module. Delta module can alter core module at class level, structure class level, interface level, or even import and export a module [3]. During realization, the entire module related should be observed to opt which module will be altered. Therefore, more efficient reusable components can be achieved.

5.1. Class Diagram

Class diagram is designed based on the component diagram in domain design. It uses UML-DOP Profile that enables the UML diagram to capture software product line engineering. It represents the core and delta modules and how those modules interacts with each other. Design of class diagram in this work follows two steps. First, designs the core module and its connection to others. Second, designs the delta modules to define modifications for core modules.

In single development, the implementation of a new feature with commonalities needs to be copied and modified in different project folder. Otherwise, delta modules make the implementation of different feature effortless with only implementing the different of the desired component.

5.1.1. Core Modules. This part shows how to design the core module and its connection to others. The design of core modules follow the same way as a single system. As shown in figure 6, it depicts a snippet class diagram of core module feature demand management.

The MDemandManagement module is a part of core module of the demand management feature as part of demand management core component. This module contains the DemandManagement class which implements the DemandManagement interface. The class represents the business logic of demand management from the OrderBased feature. The business logic is implemented with execute and calculatePIR functions.

In class diagram, the detail of components interaction are clearer with definitions which classes or modules related to each other. Based on the component diagram, SalesOrder component connected to core component feature DemandManagement. MOrderDetail is a part of SalesOrder component import MDemandManagement as part of core component feature DemandManagement.
The MPPlannedRequirement module also a part of demand management core component. It contains class PlannedRequirementImpl that implements the interface of PlannedRequirement. This class is a representation of the PIR with the attributes of the material, the number of quantity, and the period of the PIR.

![Class Diagram of Core Component Feature DemandManagement](image)

**Figure 6.** Class Diagram of Core Component Feature DemandManagement

5.1.2. Delta Modules. This part shows how to design the delta modules to define modifications for core modules. The modifications of core modules includes adds, modifies, removes classes or interfaces; adds, modifies, removes functions or types; even adds an imports or exports. Plugin components are only includes a delta modules. The modifications in delta modules derives a different products of product family and those are documented very well. Thus, it helps architect to develop new products with proper reusable components. Figure 7 depicts a snippet class diagram of core module feature demand management.

DStockDemandManagement which is a module of The DemandStrategy Plugin component defines modification of MDemandManagement as the delta module of stock based variant. The deltas modifies the MDemandManagement module represented by uses stereotype. The deltas changes the DemandManagementImpl class by changing the execute function which is marked with the modifiedClass and modifies stereotypes. Delta adds imports to the MMasterMaterial module to check the stock of materials.
Recall the concept of flexibility on plugin components, when there are new variants in a variation point, it only needs to add new variants without affects other components. In the class diagram, it applies too. The detail how the solution of a new variants only need to added, and solutions of other variants are not affected in class diagram level.

![Class Diagram of DemandStrategy Component Plugin](image)

**Figure 7.** Class Diagram of DemandStrategy Component Plugin

The concept of maintainability also applies in detailed level of class diagram. When new variation points are introduced and new component plugins are created, a new class diagram can only created for the new component plugin related. It does not affect other variation point solutions.

The differentiation of core component and plugin component and its solution to variation point in class diagram also support flexibility and maintainability. Its new solution does not affect the core component, as those are different component. The solution of its variant also does not need to affect component diagram, as the higher level of architecture.

### 6. Implementation

The implementation of SPLE in ABS Microservices are divided into different layers, such as model and resource. In the model layer, it defines the implementation of data related to demand management. Otherwise, resource layer include the business process and handle the request. The implementation are based on class diagram in subsection 5.1. This work implements core modules first as shown in figure 8, then delta modules that reuse the core modules. For the implementation of core modules are the same as implementation of a single system, while delta modules introduces new syntax declaration.

```java
module MDemandManagementResource;
export *
import OrderDetail, OrderDetailImpl from MOrderDetailModel;
interface DemandManagementResource{
  Unit execute(OrderDetail orderDetail, Int dif)
}
Unit calculatePIR(OrderDetail orderDetail, Int dif)
}
class DemandManagementResourceImpl implements DemandManagementResource{
  Unit execute(OrderDetail orderDetail, Int dif){
    Unit this.calculatePIR(orderDetail, dif);
  
  
  Unit calculatePIR(OrderDetail orderDetail, Int dif){
    // the implementation of requirement calculation
  }
}
```

**Figure 8.** Implementation of Core Modules
Delta modules define the modification of core modules. Based on the class diagram in figure 7, DStockDemandManagement only modifies the execute function of MDemandManagement. It modifies the business logic. Thus, the deltas modify the resource layer in ABS Microservices. Implementation of DDemandManagementResource is shown in figure 9.

```java
1 delta DDemandManagementResource;
2 uses MDemandManagementResource;
3 modifies class DemandManagementResourceImpl
4 modifies Unit execute(OrderDetail orderDetail, Int dif)
5 // the implementation of modification on execute
6 }
7 }
```

**Figure 9.** Implementation of Delta Modules

This implementation is based on figure 7 and its translation of stereotypes is straightforward for code implementations. The definition of delta module with delta stereotype can be seen at line 1 with Delta declaration followed by its module name. The implementation of stereotype uses is define at line 2 with uses declaration and followed by core module name. The delta module modifies class of core component at line 4 with declaration of modifies followed by the definition of its modified class. The modification on the class is defined at line 5 with the declaration modifies followed by the definition of its modified function.

7. **Evaluation**

This work shows that the architectural design and realisation of demand management of manufacturer can be carried out with the software product line engineering approach and the delta oriented programming paradigm. This design involves the variation of software product as purposed by the SPLE approach.

The designs developed in this study are Orthogonal Variability Models and feature model UML-DOP representation to model variability; package diagram that represents the system in a higher view by including variability in the related package; component diagram that explains parts of a subsystem by including variability in related parts; and class diagrams that explain the relationship between classes, modules, and their relationship with variability. The design support flexibility and maintainability of the architecture.

UML-DOP Profile and OVM enables the architecture to design the complete requirements of Software Product Line with Delta Oriented Programming into reusable artefacts software product line. UML-DOP profile enables the use of UML Diagram in the architecture. Otherwise, the architecture captures the variability across the architecture using OVM.

8. **Conclusion**

The design phase has succeeded to identify and portray commonality and variability of SPLE product. Package diagrams can illustrate how connection between features on an interrelated system. Component diagrams show how the connection between features is realized in the form of component plugins and interfaces. Component plugins and interfaces allow each component to run independently and ensure connectivity between components can be carried out. Class diagrams translate component diagrams into a structure that can be transformed into program code which is reusable artefacts for building software products. The design helps the developer in understanding the system and its relationship with SPLE elements. Using OVM and UML-DOP can simplify tracking variability in architectures with different uses. The realization phase has been successfully carried out based on the designs that have been made. The realization and design stages are linked through a feature model. The system implementation was developed using the SPLELive Engine, which includes ABS Microservices Framework [7] [8], ABS ORM [9], and OODBMS. Implementation is done through
defining features, core and delta module development as the part of domain engineering process. Then feature configuration and product selection are done as part of evaluation of the domain design and realisation process.

9. References

[1] Pohl K, Böckle G, and Van Der Linden F 2005 *Software Product Line Engineering: Foundations, Principles, and Techniques*, vol. 49, no. 12

[2] Metzger A and Pohl K 2014 Software Product Line Engineering and Variability Management: Achievements and Challenges *Future Software Engineering* pp. 70–84, 2014.

[3] Schaefer I, Bettini L, Bono V, Damiani F, Tanzarella N 2010 Delta-Oriented Programming of Software Product Lines *Lecture Notes in Computer Science*, vol 6287

[4] Setyautami M R A, Hännle R, Muschevici R, and Azurat A 2016 *A UML Profile for Delta-oriented Programming to Support Software Product Line Engineering* 20th International System Software Product Line Conference, pp. 45–49

[5] Komarudin O, Adianto D, and Azurat A 2019 Modeling Requirements of Multiple Single Products to Feature Model *Procedia Computer Science*, vol. 161, pp. 107–114

[6] Berger T et al 2013 *A survey of variability modeling in industrial practice* ACM International Conference Proceeding Series

[7] Clarke D, Muschevici R, Proença J, Schaefer I and Schlatte R 2010 *Variability modelling in the ABS language* International Symposium on Formal Methods for Components and Objects, pp. 204-224

[8] Naily M A, Setyautami M R A, Muschevici R and Azurat A 2017 *A framework for modelling variable microservices as software product lines* International Conference on Software Engineering and Formal Methods, pp. 246-261

[9] Rhizadi F, Fadhillah H S, Azurat A, Afriyanti I and Apriani N F 2019 *Database Generator to Support Product Derivation in SPL* International Conference on Advanced Computer Science and information Systems

Acknowledgement

This work is fully supported by the Reliable Software Engineering (RSE) Laboratory at the Faculty of Computer Science Universitas Indonesia as part of the PRICES research project and funded by Universitas Indonesia under Hibah PUTI Proceedings 2020 grant number: NKB-883/UN2.RST/HKP.05.00/2020