On the Benefits of Scenario Variability as Crosscutting

Rodrigo Bonifácio
Informatics Center
Federal University of Pernambuco
Recife, Brazil
rba2@cin.ufpe.br

Paulo Borba
Informatics Center
Federal University of Pernambuco
Recife, Brazil
phmb@cin.ufpe.br

Sérgio Soares
Department of Computing and Systems
University of Pernambuco
Recife, Brazil
sergio@dsc.upe.br

ABSTRACT
Variability management allows product customization by specifying variation points and composition rules related to feature models and product configurations. This is an interesting kind of crosscutting concern, since a feature might require variation points to be spread into different artifacts of each Software Product Line model (requirements, design, source code, and tests). In order to modularize use case scenario variability management, we proposed a crosscutting approach that weaves scenarios, feature models, product configurations, and configuration knowledge. The result leads to independent specification of behavior and variability concerns. In this work, we report the benefits of such kind of *separation of concerns* by comparing our approach with other techniques for handling scenario variability management.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements—*Languages, Methodologies*; D.2.13 [**Software Engineering**]: Reusable Software

General Terms
Design, Documentation

Keywords
Software product line, variability management, requirement models

1. INTRODUCTION
The Software Product Line (SPL) approach for software development corresponds to a set of practices that allows strategic reuse by observing commonalities among applications in a shared domain [6, 20]. Based on specific feature configurations, each member of the product line is assembled by selecting, creating, or customizing specific artifacts. In this way, features, that describe the SPL variability space, must be related to variation points that enable product customization. Identifying what are the variation points required for each feature, which techniques are suitable for specifying them, and how they should be related to feature models are the main goals of variability management concern. Actually, this is an interesting kind of crosscutting concern, since features might require variation points to be spread into different artifacts of SPL models.

Specifically in the context of requirement artifacts, several techniques for use case scenario variability management were proposed [14, 4, 10, 9]. However, existing works are basically concerned about how to represent variation points in scenario specification; but critical properties of variability management, such as evolvability upon common SPL increments and support for reusing specifications, are not well addressed. We argue that, due to the crosscutting characteristic of variability management, it is necessary to create a clear separation between scenario specification and variability management. Our approach (Section 2) weaves several specifications (feature model, product configuration, configuration knowledge, and use case model), which crosscut each other, to better modularize scenario variability management.

In this paper, we report on the benefits of our approach to support common increments in a real SPL (introduced in Section 3). We achieve such goal by comparing our approach to PLUC [4] and PLUSS [9], two representative notations for SPL scenario variability. In order to increase our confidence, different modularity analysis techniques were used: Design Structure Matrices (Section 4.1), a suite of metrics (Section 4.2), and observations of the effort needed to introduce SPL increments using each approach (Section 4.3). The main contributions of this paper are

- A comparative analysis that suggests the need for a better separation between variability management and SPL artifacts. Although our evaluation is restricted to use case scenario artifacts, we believe that such separation is also useful for other product line artifacts, and has already been claimed for source code [2, 8].
- An approach for comparing variability management techniques. This approach is based on techniques and common SPL increments that can also be applied to analyze modularity in other SPL artifacts. Actually, similar techniques have being used to evaluate modularity in source code [18, 11, 13].

Finally, we compare our approach to the body of related work (Section 5) and present our concluding remarks in Section 6. Next, we introduce our approach for scenario variability management.

2. VARIABILITY AS CROSSCUTTING

Variability management, as discussed in the previous section, is well known as a crosscutting concern. Therefore, in order to increase modularity, it is necessary to provide a clear separation between variability management and typical software development assets (such as requirements, design, source code, and test cases). The result is that both representations can then evolve independently. However, such kind of separation of concerns (SoC) also requires mechanisms to weave these representations during the product engineering phase [6].

We proposed a modeling framework for use case scenario variability that takes into account the crosscutting nature of variability management. A complete description of our technique is beyound the scope of this paper, but can be found elsewhere[1]. It is important to notice that our meaning of crosscutting is based on Masuhara and Kiczales work [19]. Therefore, applying the modeling framework, we described a *weaver* that takes four specifications as input: *product line use case model*, *feature model*, *product configuration*, and *configuration knowledge*. Such specifications crosscut each other with respect to the resulting product specific use case model (Figure 1). The weaver presents a clear separation between variability management and scenario specification, and also represents the processes required to compose those input specifications. Section 3, which describes a running example, presents instances of the aforementioned models.

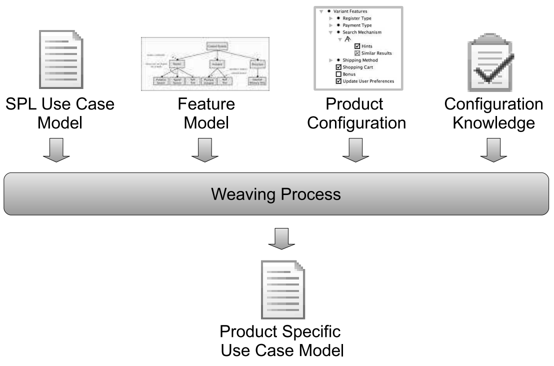

Figure 1: Overview of the weaving process

It is interesting to enforce that such composition allows the representation of different kinds of variability, such as *optional use cases and scenarios*, *quantified changed scenarios*, and *parameterized scenarios*. Additionally, the weaver was decomposed in subprocesses[1], one for each kind of variability. The semantics of those weavers (and the meta-model of the input and output languages) were described using the Haskell programming language. This decision was made in order to formally represent our weavers and to keep a similar level of abstraction that was used in [19].

[1]In fact, each subprocess is also a weaver represented in our modeling framework.

The first weaver is responsible for checking if a product configuration is a valid instance of the feature model and for selecting, based on the product configuration, the set of use cases and scenarios that should be assembled into the final product. The second weaver, instead, is responsible for composing the resulting scenarios of first weaver, since they might crosscut each other based on *from steps* and *to steps* clauses (Figure 2). These clauses are used to modularize specifications, in such way that one scenario can crosscut scenarios of different use cases.

Use Case: Create and Send a Message
Id: 01
Goal: Create and send a multimedia message
Main Flow: The message box is not full.
From step: START
To step: END

id	User Action	System State	System Response
1M	Starts the message center application		The message center application is started
2M	Selects Create New MMS option		The create a new MMS form is displayed
3M	Fills in the message body		The message body is filled in
4M	Selects the send message option		The recipient form is displayed
5M	Fills in the recipient field		The recipient field is filled
6M	Selects the confirm option	The message has not reached <FolderSize> messages	The message is sent to the recipient and saved in the sent folder. The application returns to message center

Alternative Flow: Include a multimedia content
From step: 3M,4M
To step: 4M

id	User Action	System State	System Response
A1.1	Select the include a <ContentType> option		The list of all <ContentType> available content type is displayed
A1.2	Selects a specific resource		The selected resource is attached to the multimedia message

Other alternative scenarios should be specified here

Use Case: Composition Mode Related to DRM Restrictions
Id: 02
Goal: Changes the composition behavior depended on DRM configuration
Alternative Flow: Insert a DRM restricted content in warning mode
From step: UC01/A1.1
To step: UC01/4M

id	User Action	System State	System Response
CM.1	The subscriber selects one DRM restricted resource		Request a confirmation of sending a DRM restricted content
CM.2	Confirms the attachment operation		The selected resource is attached to the multimedia message

Figure 2: Examples of scenario composition

For instance, Figure 2 depicts that the behavior of *Insert a DRM restricted content in warning mode* crosscuts the behavior of *Include a multimedia content* after Step *A1.1* (*from step* clause) and then returns to Step *4M* (*to step clause*) of *Create and send a multimedia message*. As a consequence, after weaving the scenarios showed in Figure 2, the following sequence of events is valid:

$$\boxed{< 1M, 2M, 3M, A1.1, CM.1, CM.2, 4M, 5M, 6M >}$$

The third weaver is responsible for binding parameters embedded in scenario specifications. For example, the *Include a multimedia content* scenario in Figure 2 is parameterized according to the content types supported by a specific product. The domain values of parameters are defined in the product configuration (the selected features of a SPL member).

Therefore, if a product is configured with supporting for image and audio content types, only these options will be available. In order to reduce coupling, a separated mapping is used for relating parameters embedded in scenarios to features selected in specific configurations. This decision preserves the independence between these artifacts. Although in this section we have already presented examples of our approach, the SPL used in our comparative analysis is described only in what follows.

3. MMS PRODUCT LINE

In order to compare the benefits of a clear separation between variability management and scenario specification, as supported by the approach just introduced, we specify using three different approaches, several use cases of a real multimedia message (MMS) product line. The core use cases of the MMS product line allow a subscriber to create, send and receive multimedia messages. Such messages might include both text and multimedia content (images, audio, video, rich text). Depending on specific feature configurations (Figure 3), the behavior of core use cases might change, additional use cases (like Folder Management, Configure Composition Model) might be included, and parameterized values might be instantiated. Next, we present a brief description of the MMS product line functional requirements.

Create and send a message: mandatory requirement that allows a subscriber to create and send a message. All sent messages must be saved on *sent folder*, unless the maximum number of messages (expressed by the *Size* feature) has been reached. Depending on the supported content, the user might include different kinds of objects (image, audio, video) into the message.

Receive a message: mandatory scenario that allows a subscriber to receive and view incoming messages. All received messages must be saved in the *incoming folder*. Depending on the supported operations related to structured data (phone number, email), several optional scenarios can be started (start a call, send an email, store phonebook data).

Display existing messages: mandatory scenario that allows a subscriber to navigate in the message folders and select an existing message for displaying its contents. After selecting an existing message, other operations are available, for example: edit the message contents, forward, remove, or move the selected message to a user defined folder.

Configure composition mode: optional scenario that allows a user to change the behavior of message composition with DRM (Digital Rights Management) restrict contents. If the restricted mode is selected, a message with restrict contents could not be sent. On the other hand, if the warning mode is selected, a warning will be presented if the subscriber tries to send a message with restricted content.

Structured Data Operations: optional scenario (zero or more *Structured Data Operations* can be selected) that allows a user to perform different operations from a displaying message, if the body of the message includes some structured data (like e-mail or phone number). Examples of such operations are: store a number in address book and send a message for a specific e-mail.

Based on the feature model depicted in Figure 3, and considering that the *audio* feature implies the *image* feature and the *video* feature implies the *audio* feature, we are able to derive more than one hundred members from MMS product line. The *implies* relation means that, if a product was configured with the *audio* feature, it must also be configured with the *image* feature. As a brief introduction about the feature model notation [7, 12], the relationships between a parent feature and its children are categorized as: **Optional** (features that might not be select in a specific product; e.g. *Composition Mode* and *Structured Data Operations*), **Mandatory** (features that must be selected, if the parent is also selected; e.g. *Storage Size*), **Or** (one or more sub-features might be selected; e.g. *Content Type*), and **Alternative** (exactly one sub-feature must be selected; e.g. *Composition Mode*).

For simplicity, let us assume that three products (Table 1) were defined in the SPL. This is required because some tags in PLUC technique, which will be explained later, are computed based on the specific members of the product line.

Table 1: Initial set of products

Product	Product Configuration
P1	Content Type (Image) **and** Folder Management (Size (1000))
P2	Content Type (Image, Audio, Video) **and** Structured Data Operations (Store) **and** Folder Management (Size (1000))
P3	Content Type (Image, Audio, Video) **and** Structured Data Operations (Store, Send Email) **and** Composition Mode **and** Folder Management (Size (1500), Multiple Folder)

The first configuration corresponds to a simple product, with support only for image content, maximum number of messages equals to one thousand, and no support for structured data operations and DRM composition mode. The second one has support for all kinds of contents, the same maximum number of messages (one thousand), and full support for structured data operations. The later one also has support for all kinds of contents, extended storage size (maximum number of messages equals to one thousand and five hundreds), full support for structured data operations, support for DRM composition mode and Multiple Folders. Aiming at generating the products showed in Table 1, it is necessary to relate features to software artifacts. The configuration knowledge [7, 20], one of the input models of our weaver, is responsible for keeping this kind of relation. For instance, considering the configurations described in Table 2, if the feature *Warning Mode Composition* was selected for a specific product, the scenarios *Include a Multimedia Content* (SC02) and *Insert a DRM Restricted Content in Warning Mode* (SC14) would be assembled. In a similar way, if the features *Move Message* and *Multiple Folder* were selected, the scenario *Move an Existing Message* (SC09) would be assembled.

Table 2: Segment of the Configuration Knowledge

Feature expression		Selected artifacts
Warning Mode Composition	→	(SC02, SC14)
Move Message and Multiple Folder	→	SC09
. . .		

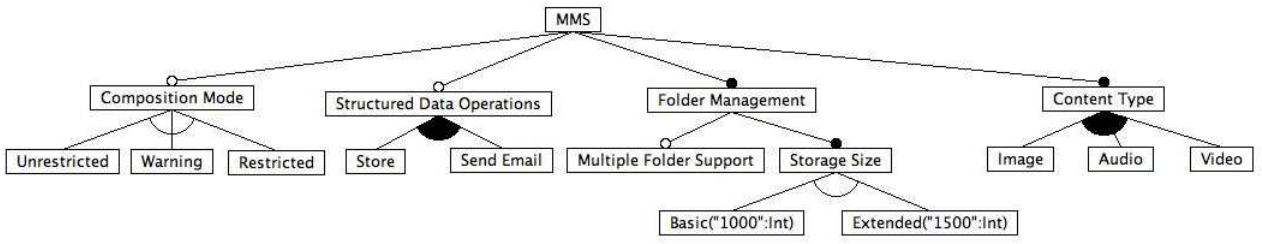

Figure 3: MMS feature model

Next, we will present the specification of the *Crate and Send a Message* use case using PLUC and PLUSS techniques. The necessary description of both notations is also presented.

3.1 Scenario specification in PLUC

Product Line Use Cases (PLUC) is a technique proposed for representing SPL requirements [4]. For instance, Figure 4 presents the PLUC specification of *Create and Send a Message.* This artifact is responsible for describing the behavior related to create and send a message and all of its extensions. In Figure 4, only two extensions are presented: *Include a Multimedia Content* and *Include a DRM Restricted Content in Restricted Mode.* Due to size constraints, other scenarios are not depicted in Figure 4.

PLUC introduces special tags for representing variabilities in use case scenarios. For example, Tag [**VP3**], in Step 6.a1 of Figure 4, denotes a variation point that abstracts over the supported content types. For each variation point used in scenario specification, one tag must be defined. The actual value of each tag is specified in the *Variation Points* section and depends on each product specification. It is important to notice that SPL members are also described using the same tag notation (see the **VP1** tag in Figure 4).

In PLUC, there is no specific artifact used to relate product configurations to feature models. In the current example, three products (Product 1, Product 2, and Product 3) are defined based on the initial set of products (Table 1). Since the values of alternative and optional variation points are computed based on such products (as can be seen from the case-of clause in Figure 4), instead of specific features, the inclusion of a new member in the product line might require a deep review of all variation points. Moreover, since the variation points and the product definitions are spread among several PLUC artifacts, preserving the SPL consistency is hard and time consuming. A systematic analysis of such properties is described in Section 4

Create and Send a Message
Goal: Create and send a multimedia message
Primary actor: Subscriber

Main Flow: The message box has not reached [**VP2**] messages.
1. The subscriber starts the message center application
2. The message center application is started
3. The subscriber selects Create New MMS option
4. The create a new MMS form is displayed
5. The subscriber fills in the message body
6. The subscriber selects the send message option
7. The recipient form is displayed
8. The subscriber fills in the recipient field
9. The recipient field is filled
10. The subscriber selects the confirm option
11. The message is sent to the recipient
12. The message is saved in the sent folder
13. The send message transient is displayed
14. The application returns to message center

Extensions

6 Include a multimedia content
6a.1.The subscriber selects the include a [**VP3**] content option
6a.2.The list of all [**VP3**] available resources is displayed
6a.3.The subscriber selects one specific resource
6a.4.The selected resource is attached to the multimedia message
6a.5.The application returns to step 6

6 Include a DRM restricted content in Restricted Mode
6b.1.The subscriber selects the include a [**VP3**] content option
6b.2.The list of all [VP3] available resources is displayed
6b.3.The subscriber selects one DRM restricted resource
6b.4.The system reports the DRM restriction
6b.5.The application returns to step 6

Other extensions should be described here

Variation points:

VP1: Alternative
0 : Product 1
1 : Product 2
2 : Product 3
VP2: Parametric
case VP1 of
0 : 1000
1 : 1000
2 : 1500
VP3: Parametric
case VP1 of
0 : (Image)
1 : (Image, Audio, Video)
2 : (Image, Audio, Video)

Figure 4: ***Create and Send a Message*** **in PLUC**

3.2 Scenario specification in PLUSS

PLUSS (Product Line Use case modeling for Systems and Software engineering) is the other technique used in our comparative analysis. This approach presents a better separation between variability management and scenario specification, since product definition is not tangled within use cases, and the domain values of parameters (similar to PLUC tags) are related to alternative features and are not embedded in specifications [9]. However, according to PLUSS metamodel [9], there is no specific artifact (such as configuration knowledge [7, 20]) used for relating features to optional use cases, scenarios and steps. Another characteristic of PLUSS is that all variant steps of a scenario specification are defined in the same flow of events. For example, the third flow presented in Figure 5 describes the behavior related to the attachment of a DRM restricted content when a product is configured in both *restricted* or *warning* mode. Steps 2(a) and 2(b) are never performed together. They are alternative steps: Step 2(a) will be performed only if the feature *Restricted composition mode* is selected; otherwise Step 2(b) will be performed. Additionally, Step 3 is optional, and will be performed only if the feature *Warning composition mode* is selected. Therefore, based on PLUSS notation, features must be directly related to the use case model.

In the current example, it is necessary to annotated the PLUSS feature model in order to represent that the *Storage Size* feature is related to the **Size** parameter; the *Content Type* feature is related to the **ContentType** parameter; the *Restricted composition mode* feature is related to the Step 2(a) of second extension; and the *Warning composition mode* feature is related to steps 2(b) and 3 of the second extension. Finally, notice that the behavior described in the second extension of Figure 5 might be required in other use cases, like *Selecting a Message to Forward* - since it is possible to attach a multimedia content while forwarding a message. Hence, a feature behavior can be spread among several use cases, resulting in maintainability issues: introducing a new product variant might require changes in several artifacts.

Create and Send a Message
Goal: Create and send a multimedia message
Primary actor: Subscriber
Main Flow: The message box has not reached [**SIZE**] messages.

id	Actor Action	System Response
1	Starts the message center application	The message center application is started
2	Selects Create New MMS option	The create a new MMS form is displayed
3	Fills in the message body and selects the send message option	The recipient form is displayed
4	Fills in the recipient field	The recipient field is filled
5	Selects the confirm option	The message is sent to the recipient and saved in the sent folder. The application returns to message center

Extension Flow: Include a multimedia content

Id	Actor Action	System Response
1	In the third step, the subscriber selects the include a [**ContentType**] content option	The list of all [**ContentType**] available resources is displayed
2	Selects one specific resource	The selected resource is attached to the multimedia message and the application returns to step 6

Extension Flow: Include a DRM restricted content

id	Actor Action	System Response
1	Selects the include a [**ContentType**] content option	The list of all [**ContentType**] available resources is displayed
2(a)	Selects one DRM restricted resource	The system reports the DRM restriction: the subscriber can not attach the resource
2(b)	Selects one DRM restricted resource	The system asks the subscriber if he (she) wants to send a DRM restricted content
(3)	Confirms the attachment operation	The selected resource is attached to the multimedia message

Figure 5: *Create and Send a Message* in PLUSS

In what follows, we present a systematic evaluation of our crosscutting approach by comparing it with the techniques just presented.

4. EVALUATION

As briefly discussed in Section 3.1 and Section 3.2, both PLUC and PLUSS approaches do not present a clear separation between variability management concern and scenario specification. This occurs because variants in PLUC are embedded in use case specifications; and, although variants in PLUSS are not tangled within use cases, such technique requires a direct relation from features to use cases. As a result of this kind of dependency, its is difficult to evolve both representations independently. In this section, we apply Design Structure Matrices (DSMs) [3] for better understanding the effects of such dependences. DSMs is an interesting and simple tool for visualizing dependences between design decisions. Such decisions are distributed in both rows and columns of a matrix.

We can identify which input data is required (a dependency) by a design task by observing which columns are marked in its corresponding row. However, DSMs do not offer support for quantifying relevant properties of scenario variability management. Therefore, we also present an evaluation based on modularity and complexity metrics. Finally, in order to increase the confidence of our previously evaluations, we analyze how the compared approaches support some common SPL increments, such as introducing a new feature variant and introducing a new product configuration.

4.1 DSMs analysis

In this work, we apply DSMs to visualize design dependences in two levels: the first one presents a high level view of dependences between variability management (feature model, SPL instances and configuration model) and use cases; the second one presents how features are spread among use cases. Observing the high level view of dependences in PLUC (Figure 6(a)), we can realize that such technique results in cyclical dependences between use cases and variability management. First, use cases refer to tags that are computed based on features and on specific SPL instances (dependences illustrated in line 4 with columns 1, 2, and 3). On the other hand, the relationship between features and artifacts and the definition of SPL instances are embedded in the use cases (dependences illustrated in lines 1, 2, and 3 with column 4). This is an example of a non-modular design, because, in PLUC, the *Create and Send a Message* use case (Figure 4) specifies the space variability for the *Message Folder Size* and the *Supported Content Types* features. However, the same information is presented in the Receive a Message use case. Additionally, the SPL valid instances (products) and its configurations are spread among several use cases. These dependences result in several modularity issues, being difficult for: a) documenting variability management and use case decisions in parallel; and b) evolving both representations independently.

(a)

		1	2	3	4
1	Feature model				x
2	SPL instances	x			x
3	Configuration model	x	x		x
4	Use case model	x	x	x	

(b)

		1	2	3	4
1	Feature model			x	x
2	SPL instances	x			
3	Configuration model	x			x
4	Use case model	x			

(c)

		1	2	3	4	5
1	Feature model					
2	SPL instances	x				
3	Configuration model	x				x
4	Environment	x				
5	Use case model				x	

Figure 6: High level view of dependences

PLUSS partially solves (see Figure 6(b)) the cyclical dependences just presented, since SPL instances and feature variants are not embedded in use cases. However, we can visualize a cyclical dependency between the feature model and the configuration knowledge, since there is no independent artifact used to relate features to use cases, Additionally, it is not possible to independently evolve parameters in use case specification and their domain values defined in feature models (resulting in the cyclical dependency between use cases and features). Applying our approach results in a better separation (Figure 6(c)) between variability management and use cases, since the relationships between them are specified in the configuration knowledge (they are not established in the feature model, as in PLUSS).

As explained before, in PLUSS, it is necessary to introduce some annotations in the feature model in order to define the domain values of a scenario parameter. On the other hand, in our approach, we introduced a third artifact (a map or environment) responsible for this kind of relationship. Such map is composed by pairs *Parameter name* x *Feature id.* As a consequence of introducing such mapping, the name of features can change without breaking the use cases (only the environment should be updated). Because use cases do not have explicit references to SPL instances and configurations, introducing new features or SPL instances do not change the use case model.

In the context of this work, DSMs were also applied for representing the spreading of features into use cases. Such representation was very useful for computing the metric *Feature Diffusion Over Use Cases* [11] (explained in Section 4.2). Due to space constraints, we do not present the PLUSS DSM in this level of details. We can observe in the DSM of Figure 7(a) that, in PLUC, features are often spread among several use cases. For example, feature *Composition Mode* is relevant in the context of four use cases (labeled as the design parameters 5 to 8). Other interesting characteristic, which we can observe in the Figure 7(a), is that there is no dependences between use cases, since each use case describes all of its extension points. However, it is not possible to specify two use cases independently, since there is cyclical dependences between use cases and features and these features can be spread into several use cases.

In our approach, since one scenario can refer to steps specified in another use cases (either using *step ids* or *annotations*), there are several dependences between use cases. For example, in Figure 7(b), *Structured Data Operation* use case depends on *Select and Display a Message* and on *Receive a Message* use cases. However, features do not depend on use cases anymore (the variability space is not represented in use cases), there is no cyclical dependences, and use cases are more concise (although the number of use cases is greater than in PLUC). Two new use cases were created: *Forward a Message*, that allows the reuse related to forward an existing message; and *Structured Data Operations*, responsible for modularizing all specification related to the structured data operation concern.

4.2 Quantitative analysis

Based on the scenario specifications and on the DSM analysis (Section 4.1), we derived several metrics for quantifying feature modularity and use case model complexity (related to the size of specifications).

(a)

	1	2	3	4	5	6	7	8	9
1 Composition mode					x	x	x	x	
2 Structured data operations						x	x		
3 Folder management					x	x			x
4 Content type					x	x			
5 Create and send a message	x		x	x					
6 Receive a message	x	x	x	x					
7 Select and display a message	x	x							
8 Configure composition mode	x								
9 Manage user defined folders			x						

(b)

	1	2	3	4	5	6	7	8	9
1 Composition mode									
2 Structured data operations									
3 Folder management									
4 Content type									
5 Create and send a message			x	x					
6 Receive a message			x	x					
7 Select and display a message				x					
8 Configure composition mode	x								
9 Manage user defined folders			x						
10 Forward a message						x	x		
11 Structured data operation		x				x	x		

Figure 7: Features and use case dependences

Increasing levels of feature modularity implies better evolvability, since SPL changes or increments can be performed in a isolated way. Also, if a modular feature specification could crosscut other specifications, it would be expected more reusable assets.

Adapted from [13], the proposed modularity metrics quantify two types of relations involving features and use cases. First, *Feature Diffusion over Use Cases* (FDU) is used for quantifying how many use cases are affected by a specific feature. For instance, in PLUC we can realize (Figure 7(a)) that *Folder Management* feature affects *Create and Send a Message* and *Receive a Message* use cases. So, the corresponding FDU value is equal to two. On the other hand, *Number of Features per Use Case* (NFU) is used for quantifying how many features are tangled within a specific use case. We assume that each use case should be interested in its primary goal, although several features might be related to the primary goal of a use case. For example, *Content Type* and *Storage Size* features are part of the primary goal of *Create and Send a Message* use case. However, *Composition Mode* and *Structured Data Operations* do not compose its primary goal - actually, a specific product can be assembled without such features. Therefore, NFU value for *Create and Send a Message* PLUC is three. Moreover, we applied the metric *Feature Diffusion over Scenarios* (FDS) in order to quantify how many internal use case members (scenarios) are necessary for the materialization of a specific feature.

Two metrics related to complexity were applied in this work. The first one, vocabulary size, quantifies the number of use cases (VS_U) and scenarios (VS_S) required by each of evaluated approaches. The second one, *Steps of Specification* (SS), is related to the size of each scenario and identifies how many pairs *User action* x *System response* compose a specific scenario. Additionally, we also relate modularity to complexity by applying *Features and Steps of Specification* (FSS), which counts the number of steps of specification whose main purpose is to describe the behavior of a feature.

Table 3 summarizes the evaluation of these metrics based on the case study.

Table 3: Modularity and complexity metrics

	PLUC	PLUSS	Crosscutting
Mean value of FDU	3.5	3.5	2
Mean value of FDS	6.25	5	4.25
Mean value of NFU	2	2	1
Mean value of FSS	12	11	10.25
VSU	5	5	7
VSS	27	24	23
SS	75	64	56

In Table 3 we present the average value of some metrics, such as the mean value of *Feature Diffusion over Use Cases* (FDU). Therefore, on the average, the running example points that that each feature in PLUC is tangled within 3.5 use cases. Notice that, since PLUC and PLUSS do not allow a scenario to crosscut other scenarios in different use cases, it is difficult to modularize features into a single use cases. The result is that, when comparing to the crosscutting approach, features are more diffused (FDU metric) and use cases are less concise (NFU) in these approaches. The crosscutting approach, in contrast, allows the composition of scenarios through *from steps* and *to steps* clauses. Such mechanism, although more expressive and formal, is similar to use case extensions [15, 16, 17].

Another point is that *vocabulary size metrics* (such as VSU and VSS) are better in the crosscutting and PLUSS approaches for different reasons. First, the number of scenarios in PLUSS is lower than in PLUC because a scenario in PLUSS might be used for describing all related variants. For instance, *Include a DRM Restricted Content* scenario of Figure 5 describes the behavior for both *Restricted* and *Warning* modes. This facility is not supported in PLUC neither in crosscutting approach. However, in the crosscutting approach the vocabulary size is lower than in other approaches for the reason that a scenario might be modularized in such way that no duplications are required. For example, in PLUC and PLUSS approaches, since *forward an incoming message* operation can be started from *Receive a Message* and from *Select and Display a Message* use cases, scenarios related to DRM constraints are duplicated. This problem can be avoided in our crosscutting approach by composing scenarios of different use cases. Also related to vocabulary size, the crosscutting approach presents more use cases than PLUC and PLUSS. However, the number and complexity of scenarios are lower. We discuss a bit more about these conclusions in Section 4.4. Next, we present the last evaluation we performed in this work.

4.3 SPL evolution analysis

This section aims at evaluating how the results of DSMs and quantitative analysis may be related to the flexibility, which is quantified by introducing some common SPL increments. The new version of MMS product line introduces a new *structured data operation*, allowing a subscriber to make a call for a number embedded in a message; introduces a new *content type*, allowing a subscriber to attach emotion icons to messages; and defines a new product with the configuration presented by Table 4.

Table 4: New MMS product line member

Product	Feature Configuration
P4	Content Type (Image, Audio, Emotion icons) Data Operations (Store, Send Email, Place a call) Composition Mode Folder Management (Size (1000), Multiple Folder)

Table 5 summarizes the changes required by each SPL increment. We evaluate, for each technique, the impact of a given increment on the following SPL artifacts: feature model (FM), configuration knowledge (CK), product configurations (PC), and use case model (UC). Specifically for the use case model, we present how many artifacts were impacted. For the other ones, we present only if the artifact was modified or not. The main conclusion is that defining domain values for parameters tangled within use cases requires a lot of effort to make simple changes, such as the inclusion of a new content type. In PLUC, this increment requires changes in all models. Additionally, each use case that refers to the content type must be updated. Both PLUSS and crosscutting approaches, which use feature models to define the domain values of parameters, require only changes in feature models to introduce a new feature variant (a new content type, in this case). Based on the increments proposed, there is no significant difference between PLUSS and the crosscutting approach. However, since the behavior related to *Strucure Data Operations* are modularized in one use case, in the first increment, the number of affected use cases in the crosscutting approach is lower than in PLUSS.

4.4 Threats to conclusion validity

We have chosen the MMS product line because it offers the opportunity of specifying different kinds of variability, such as *optional use cases and scenarios*, *parameterized scenarios* and *crosscutting changes*. However, we concluded that some differences in the quantitative analysis might be observed after evaluating other product lines. For instance, besides some issues related to understandability, combining different variants in a single scenario might reduce significantly the specification's vocabulary size. This characteristic is only supported by the PLUSS approach. However, in the running example, only one feature was suitable for being specified using this technique. Additionally, we have specified the PLUC and PLUSS scenarios based on a few available examples. All scenario specifications are available at [1]. However, the evaluation discussed in Section 4 is still valid; once it can be used as a guide to perform other analysis and to report the benefits of the SoC between variability management and SPL assets.

5. RELATED WORK

Several approaches have been proposed for representing scenario variability [16, 14, 9, 4]. However, in this paper we only compare our crosscutting approach with PLUC and PLUSS techniques because they encompass a broad range of SoC between variability management and scenario specification. PLUC presents the lowest level of modularity, since almost all information related to variability is tangled within use cases. Although PLUSS partially reduces such coupling, by considering the importance of feature modeling, some dependences from feature to use cases are still present. These dependences are avoided in our approach.

Table 5: Changes required in SPL increments

	PLUC				PLUSS				Crosscutting			
Increment	FM	CK	PC	UC	FM	CK	PC	UC	FM	CK	PC	UC
New data operation	x	x	x	2	x	x		2	x	x		1
New content type	x	x	x	3	x			0	x			0
New product definition	x	x	x	3			x	0			x	0

Our weaver for scenario compositions relies on references to *step ids* or *step annotations.* The use of annotations is an attempt to reduce the problem of fragile pointcuts. A more reliable approach is presented in [5], which describes a composition mechanisms based on natural language processing. However, it is possible to introduce this technique by implementing a new matching function that is used to retrieve steps referred by *from steps* and *to steps* clauses.

Design Structure Matrices (DSMs) and crosscutting metrics have being applied for assessing modularity in *aspect-oriented* systems [18, 11, 13]. In this work, we also applied DSMs in the context of scenario variability management. Moreover, we customized a suite of metrics for quantifying feature diffusion and tangling over use cases. To our knowledge, our work is unique in applying both DSMs and crosscutting metrics for evaluating SoC in scenario variability.

6. CONCLUDING REMARKS

Variability management is a common challenge in software product line adoption. Although several works have been proposed for managing variabilities at scenario specification, existing works focus on representing variation points at use case documentation. In this paper we reported that we also need to introduce a modular perspective in this domain. On the reason that, if variability management is tangled within scenario specifications, it will be difficult to evolve both representations . Our analysis were based on customizations of recent techniques for software modularity assessments, such as *Design Structure Matrices* and crosscutting concerns metrics. We showed that, in order to introduce common SPL increments, a clear separation between variability management and use case specifications is extremely valuable. As future work, we aim at evaluating other product lines and improve the presented metric suite.

7. REFERENCES

[1] Software productivity group, online: http://www.cin.ufpe.br/spg (2007).

[2] V. Alves, A. C. Neto, S. Soares, G. Santos, F. Calheiros, V. Nepomuceno, D. Pires, J. Leal, and P. Borba. From conditional compilation to aspects: A case study in software product lines migration. In *GPCE'06*, Portland, USA, 2006.

[3] C. Baldwin and K. Clark. *Design Rules The Power of Modularity*, volume 1. The MIT Press, first edition edition, 2000.

[4] A. Bertolino and S. Gnesi. Use case-based testing of product lines. In *ESEC/FSE'03*, pages 355–358, New York, NY, USA, 2003. ACM Press.

[5] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters. Semantics-based composition for aspect-oriented requirements engineering. In *AOSD '07*, pages 36–48, New York, NY, USA, 2007. ACM Press.

[6] P. Clements and L. Northrop. *Software product lines: practices and patterns.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[7] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley Professional, 2000.

[8] M. de Medeiros Ribeiro, P. M. Jr., P. Borba, and I. Cardim. On the modularity of aspect-oriented and other techniques for implementing product lines variabilities. In *First LA-WASP*, João Pessoa, Brazil, oct 2007.

[9] M. Eriksson, J. Borstler, and K. Borg. The pluss approach - domain modeling with features, use cases and use case realizations. In *SPLC'05*, 2005.

[10] A. Fantechi, S. Gnesi, G. Lami, and E. Nesti. A methodology for the derivation and verification of use cases for product lines. In *SPLC'04*, 2004.

[11] A. Garcia, C. SantAnna, E. Figueiredo, U. Kulesza, and C. Lucena. Modularizing design patterns with aspects: A quantitative study. In *Transactions on Aspect-Oriented Software Development.* Springer, 2005.

[12] R. Gheyi, T. Massoni, and P. Borba. A theory for feature models in alloy. In *First Alloy Workshop*, pages 71–80, Portland, United States, nov 2006.

[13] P. Greenwood, T. Bartolomei, E. Figueiredo, M. Dosea, A. Garcia, N. Cacho, C. Santanna, S. Soares, P. Borba, U. Kulesza, and A. Rashid. On the impact of aspectual decompositions on design stability: An empirical study. In *ECOOP: Proceedings of the European Conference on Object-Oriented Programming*, 2007.

[14] M. L. Griss, J. Favaro, and M. d' Alessandro. Integrating feature modeling with the RSEB. In *ICSR '98*, page 76, Washington, DC, USA, 1998. IEEE Computer Society.

[15] I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley Professional, 1992.

[16] I. Jacobson, M. Griss, and P. Jonsson. *Software reuse: architecture, process and organization for business success.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997.

[17] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases.* Addison-Wesley Professional, 2004.

[18] C. V. Lopes and S. K. Bajracharya. An analysis of modularity in aspect oriented design. In *AOSD '05*, pages 15–26, New York, NY, USA, 2005. ACM.

[19] H. Masuhara and G. Kiczales. Modeling crosscutting in aspect-oriented mechanisms. In *ECOOP'03*, 2003.

[20] K. Phol, G. Bocle, and F. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, 2005.