Assessment of Product Derivation Tools in the Evolution of Software Product Lines: An Empirical Study

Mário Torres, Uirá Kulesza, Matheus Sousa, Thais Batista
DIMAp-UFRN, Brazil
{mario, uira, thais, matheus}@dimap.ufrn.br

Leopoldo Teixeira, Paulo Borba
CIn-UFPE, Brazil
{lmt, phmb}@cin.ufpe.br

Elder Cirilo, Carlos Lucena
PUC-Rio, Brazil
{ecirilo, lucena}@inf.puc-rio.br

Rosana Braga, Paulo Masiero
ICMC-USP, Brazil
{rtvb, masiero}@icmc.usp.br

ABSTRACT

Product derivation approaches automate the customization process of software product lines. Over the last years, many tools have been proposed aiming at synthesize and generate products from a set of reusable assets. These tools adopt different techniques and strategies to implement and automate the product derivation activities. In this paper, we analyzed six modern product derivation tools (Captor, CIDE, GenArch, MSVCM, pure::variants, XVCL) in the context of evolution scenarios of a software product line. Our study has adopted several metrics to analyze the modularity, complexity and stability of product derivation artifacts related to configuration knowledge along different releases of a mobile product line. The preliminary results of our study have shown that approaches with a dedicated model or file to represent the CK specification can bring several benefits to the modularization and stability of a software product line.

Categories and Subject Descriptors

D.2.8 [**Metrics**]: Product Metrics.

General Terms

Measurement, Experimentation

Keywords

Product Derivation Tools, Measurement

1. INTRODUCTION

A software product line (SPL) [8] is a set of related software systems from a particular market segment that share common functionalities, but are sufficiently distinct from each other. Existing approaches [9, 8] propose and motivate SPL development by means of the specification, modeling and implementation of features. A feature [9] is a system property or functionality that is relevant to a stakeholder. It is used to capture commonalities and discriminate variabilities among SPL systems. SPL development involves the design and implementation of core assets (components, frameworks, libraries and others) that adequately modularize common and variable features during domain engineering [9].

Product derivation [10] refers to the process of building a product from the set of code assets implemented for a SPL. It encompasses the selection, composition and customization of these code assets, in order to address a specific SPL product (configuration). Existing product derivation approaches [7, 6, 22, 4, 20, 13, 14, 3] automate the synthesis and customization of SPL products. Over the last years, many tools have been proposed with this aim. They adopt different techniques and strategies to implement and automate the product derivation activities, varying in different perspectives, such as: (i) from visual and model-based tools to textual and domain-specific approaches that specify the problem space (e.g., feature model), solution space (e.g., code assets) and configuration knowledge (mapping between features and code assets) from the SPL; and (ii) they adopt a positive or negative derivation process to customize and generate SPL products.

Many tools have been proposed, with several advantages of their adoption in real and industrial scenarios [7, 6, 22, 4, 20, 13, 14, 3]. However, there are few studies addressing the assessment and comparison of these tools that demonstrates the real impact, benefits and disadvantages of using a specific tool. Existing research work focuses on qualitative tool analysis [18, 21, 16, 15]. None of the existing studies have explored or analyzed the product derivation artifacts produced during the evolution of an existing SPL. Besides, to the best of our knowledge, there is no existing work that quantifies metrics that assess the modularity, complexity and stability of product derivation artifacts during the evolution of SPLs.

In this context, this work proposes to assess and compare existing product derivation approaches considering the evolution releases of a SPL. Our analysis focuses mainly on the modularity, complexity, and stability attributes of derivation artifacts specified to support the process of automatic product derivation. Existing metrics adopted in other recent empirical studies [12, 19] are adapted to quantify these attributes in product derivation artifacts that specify the configuration knowledge (CK) between features and code assets. The metrics are computed in the aspect-oriented implementations of four evolution releases of MobileMedia [12], a software product line for media (photo, video and audio) management on mobile devices. The following six existing product derivation tools are analyzed and compared in our study

FOSD'10, October 10, 2010, Eindhoven, The Netherlands.

from the perspective of CK specification: Captor [20, 13], CIDE [14], GenArch [7], Hephaestus [5], pure::variants [3] and XVCL [22]. The preliminary results of our study have shown that approaches with a dedicated model or file to represent the CK specification can bring several benefits to the modularization and stability of a software product line.

The remainder of this paper is organized as follows. Section 2 presents the study settings of our comparative study: it overviews the investigated product derivation approaches, it details the phases and assessment procedures adopted, and finally, it describes the metrics adopted to quantify the modularity, complexity and stability of the SPL releases. Section 3 analyzes and discusses the results obtained for the metrics considering the different releases of MobileMedia. Some general discussions are made on section 4. Section 5 relates our study to other existing ones developed by the community. Finally, Section 6 concludes the paper and provides directions for future work.

2. STUDY SETTINGS

This section presents detailed information about our assessment of product derivation tools. The main aim of our study was to analyze and observe the modularity and stability of existing product derivation artifacts, and also to validate the usefulness of some metrics in the quantification of these attributes. In Section 2.1, we present the approaches we have evaluated in this study. Section 2.2 presents the phases and assessment procedures of our preliminary study. Section 2.3 describes the metrics suite adopted to enable the modularity, complexity and stability analysis of the approaches.

2.1 Product Derivation Approaches

This section provides an overview of the evaluated approaches. We discuss their particularities, identifying how the configuration knowledge (CK) is expressed in each approach. In most of the approaches, **feature models** [9] are used to represent the commonalities and variabilities of the SPL, defining its scope. Feature models denote the set of products that can be generated for the SPL, through the relationships between features and their types (alternative, optional, mandatory, and so on). The approaches in our preliminary study were chosen according to the following criteria: (i) they represent approaches that were developed by our research groups (Captor, MSVCM and GenArch) or they are considered relevant product derivation tools in the research or industrial community (CIDE, pure::variants and XVCL); (ii) they are code-oriented, meaning that when we refer to **reusable assets**, we are mainly referencing code assets, such as classes, aspects, interfaces, packages, except stated otherwise; and (iii) there is an available implementation of the approach in order to allow its use in the case study. Unfortunately, due to time restrictions, it was not possible to include or consider other product derivation tools in this preliminary study. We intend to extend and consider new approaches in a future and more controlled study.

Captor. The Captor tool is a Configurable Application Generator used to support the development of applications on a specific domain [20]. It covers domain and application engineering. It uses an Application Modeling Language (AML), used in a way similar to a feature model, that can be specified through a graphical user interface, or directly in XML. The CK is also composed by templates with XSL tags, and a mapping file called rules.xml, that links the defined AML with the templates. Captor also provides pre and post processors, also specified on XML files, which can be used to define tasks like copying mandatory files to the output directory. The application engineer defines an instance of the AML previously created to build and derive a product.

CIDE. The Colored Integrated Development Environment (*CIDE*) is a SPL tool for decomposing legacy code into features [14]. It follows the paradigm of virtual separation of concerns, i.e., developers do not physically extract the feature code, but just annotate code fragments inside the original code, in a similar fashion to conditional compilation tags (*#ifdef*). However, instead of using tags in comments throughout the code, it uses background colors. So, code fragments belonging to a feature are shown with the background color of the feature. Another difference to conditional compilation is that annotations are disciplined, in order to prevent syntax and type errors. The underlying structure of the code to be annotated is considered, allowing developers to annotate, and thus, remove from assets, only program elements like classes, methods, or statements. These annotations represent the CK information, associating these elements to features.

GenArch. GenArch [7, 6] is a model-based tool for automating the product derivation process. The GenArch approach is centered on the definition of three models: feature, architecture and configuration. The architecture model defines a visual representation of the reusable assets in order to relate them to features. The configuration model is responsible for defining the mapping between features and assets, representing the CK. This model is fundamental to link the problem space (features) to the solution space (implementation assets), and to enable automatic product derivation.

MSVCM. The Modeling Scenario Variability as Crosscutting Mechanisms (*MSVCM*) approach [4] was initially proposed to deal with requirements variability, but it has been extended to deal with variabilities in source code and build files [5]. In MSVCM, the CK is specified into a separate model, relating features and their combinations (feature expressions) to transformations that translate SPL assets into product specific artifacts. The approach is named crosscutting because product derivation is resultant of a weaving process that takes as input artifacts such as the feature model, configuration knowledge, instance model, and so forth. These models crosscut each other with respect to the resulting product. If a given feature expression is evaluated as *True* for a given product (defined in the instance model), the related transformations are applied.

pure::variants. pure::variants [3] is a SPL model-based product derivation tool. Its modeling approach considers mainly two models: feature and family models. The family model describes the internal structure of the individual components and their dependencies on features. The family model is structured in several levels. The highest level is formed by the components. Each component represents one or more functional features of the solutions and consists of logical parts of the software (reusable assets). The physical elements can be assets that already exist, assets that will be created and transformations that will be performed based on the feature selection. A transformation can be any activity, such as copying code assets from a repository to a specific location, customizing configuration files, or even UML

models.

XVCL. The XML-based Variant Configuration Language (*XVCL*) [22] is a language for configuring variability in textual-based assets. XVCL is based on Bassett's frames [2], every file on its structure needs to be a frame (XML file combined with code and XVCL commands) linked to other frames. This hierarchical structure is called *x-framework*. This approach has no specific structure to organize the CK. Instead, we use XVCL variables to set features and then validate if that feature should be present or not in the derivated product. The XVCL processor then, given a set of features (variables) and their values (selected or not) processes the frames files in order to generate the product.

2.2 Study Phases and Assessment Procedures

Our study was organized in the following major phases: (i) specification of the SPL artifacts related to product derivation considering all the approaches presented in Section 2.1; (ii) quantification of the selected metrics over the different derivation artifacts produced for each one of the investigated approaches; and (iii) quantitative analysis and assessment of the obtained results for the different modularity, complexity and stability metrics adopted in our study. Following we provide additional details of these phases.

In the first phase of our study, the aspect-oriented implementations of 4 releases (release 4 to 7) of MobileMedia SPL [12] were considered to implement the different artifacts of the product derivation approaches. The available documentation of MobileMedia was used as a base to specify and implement the derivation artifacts. MobileMedia (MM) was selected to be part of our study for different reasons. First, because it is an expressive SPL implemented with modern technologies, including an aspect-oriented language (AspectJ) and the Java Micro Edition (Java ME) API. Second, it has been used and validated in many other empirical studies [12, 1]. Finally, MM provides different evolution releases which allowed us to observe the effects of change scenarios along derivation artifacts considering the different approaches. This last criterion was preponderant to the choice of MM to this preliminary study.

During the specification of the derivation artifacts, we established alignment rules between the approaches in order to guarantee that: (i) the best practices of each approach were used to implement the artifacts; and (ii) the comparison between the derivation artifacts was equitable and fair. Five researchers performed these alignment activities. All misalignments found were discussed between the study participants and eventual corrections were applied to the artifacts implementation to guarantee their alignment. It was ensured, for example, that: (i) the same set of common and variable features were used in the derivation artifacts considering each release of MM; (ii) every variability and implementation artifacts were expressed using the appropriate mechanisms of the product derivation approaches; and (iii) the CK specifications in the different approaches are consistent between them, which means that all product derivation approaches are specifying the same products that can be automatically produced from each MM release.

After implementing MM derivation artifacts using the six different approaches, we applied and quantified the modularity, complexity and stability metrics along these different artifacts. We considered the artifacts that are responsible to specify the configuration knowledge in the different approaches. Our main aim was to quantify the tangling, scattering, size and instability of the derivation artifacts in the light of change scenarios demanded by MM evolution. Additional details about the adopted metrics to quantify these properties are presented in the next section. Finally, after the collection of all the metrics, the computed data was organized in spreadsheets and graphics in order to be analyzed. Results of this analysis are presented in Section 3.

2.3 The Metrics Adopted in Our Study

In order to compare the CK specification of the different product derivation approaches, we have selected a metrics suite to enable their quantitative analysis. The metrics are divided into three main groups: (i) modularity, (ii) complexity and (iii) stability.

Modularity. The modularity metrics are adapted from previously proposed metrics by Sant'Anna et al [19]. This previous work has proposed a set of modularity metrics to measure the separation of concerns in aspect-oriented implementations. The main goal of these metrics is to quantify the degree of scattering and tangling of concerns in aspect-oriented artifacts. They have been used and validated not only in the assessment of aspect-oriented implementations, but also to artifacts produced in other development stages: such as requirements, architecture and design, textual and model specifications [4,12]. In our study, we have adapted these metrics to quantify the scattering and tangling of CK specifications along product line assets (configuration files, derivation models, templates and source code) that are implemented to enable automatic product derivation.

We measure the scattering counting 1 for each SPL asset that contains some sort of CK specification, including any textual document or model associated with CK, when applied. In the CIDE approach, we counted every code asset colored according to a specific feature, as a unit of CK scattering. Tangling is calculated in a similar way than the Concern Diffusion over Lines of Code (CDLoC) metric, where we count the number of concern switches in a given source code asset [19]. In our case, we calculate this metric considering the CK specification as the concern. Thus, for each derivation artifact that includes some sort of CK specification tangled with specification or code related to another concern (variabilities and implementation), there is a switch, which we count. If the approach has a dedicated CK model or textual specification, we do not take this model into account for this metric, because it completely modularizes this concern.

Complexity. The goal of this analysis is to measure the effort needed by domain engineers to prepare the artifacts that support the automatic product derivation in a specific approach. Complexity is directly related to the size of configuration items needed to represent the CK in each approach. Two metrics were used: (i) number of tokens in CK sentence expressions; and (ii) number of CK sentence expressions.

The first metric counts the number of tokens needed to build the CK, i.e., the data that the domain engineer must effectively write to configure the SPL according to the feature model. The counting was based on the native tokens provided by each language/tool to represent the CK. In the XVCL approach, for example, we define the following expression for the feature Photo: `<select option="Photo"> <option value="yes">`. Here we count the total of 16 tokens, 8 for each statement. On the other hand, in

the GenArch tool, it is only necessary to provide the feature name associated - in this case, *Photo* - with a specific code asset in the configuration model. Thus the number of token in this case for GenArch was 1, because that is all the domain engineer has to write. The complete specification is transparent and is maintained through the configuration model.

The second metric, number of CK sentence expressions, analyzes the conciseness and expressiveness of the CK in the different approaches. It quantifies the amount of CK sentence expressions needed to support the product derivation process. When the SPL evolves, the concision of these expressions becomes even more important. In some approaches, it enables adding new assets without requiring the inclusion of a new expression. We count the number of CK sentence expressions by quantifying the amount of feature expressions specified in the CK artifacts. Note that this counting is independent of the effort to build the expression or its size. These are addressed by the number of tokens in CK sentence expressions metric.

Stability. The stability metric is used to analyze the impact of evolving the SPL on the derivation artifacts (CK). The metric was computed in terms of CK sentence expressions added, changed or removed during the SPL evolution. We measure the difference between releases, it is inspired on Yau and Collofello study [23]. A CK sentence expression is considered new when there is a new feature expression in the CK for that release. When a new code asset is included in the SPL, this not necessarily imply in the inclusion of a new CK feature expression. This depends on the approach. In some cases, an existing expression can be modified to address the new code assets. With this metric analysis we can measure the effort needed to evolve the CK during SPL maintenance.

3. STUDY RESULTS

In this section, we present and discuss the collected results for the modularity, complexity and stability metrics from our study. Our analysis considers the specification of the derivation artifacts considering 4 different releases of the Mobile Media SPL.

3.1 Modularity Analysis

The main goal of modularity analysis is to quantify the degree of scattering and tangling of the CK over the SPL assets.

Configuration Knowledge Scattering. We measured the degree of scattering of the configuration knowledge by quantifying all SPL assets that have some sort of CK specification in them. Source code, for example, might contain such information in the form of a conditional compilation tag (*#ifdef*). Figure 1 shows the results of the collected values for this metric considering the four releases of MobileMedia specified using the different approaches. We can observe that GenArch, pure::variants and MSVCM tools presented more stable and lower values for this metric compared to other approaches, even when new features and assets are added. This happens mainly because these approaches provide a separate model or configuration model to specify the configuration knowledge with the mapping between features and code assets. The CIDE, Captor, and XVCL approaches presented higher values for the CK scattering, as shown in Figure 1. CIDE presents a higher scattering because it does not provide dedicated support to modularize the CK. In this approach, every colored element in the code assets can be seen as a CK sentence expression that relates the element to a feature. The Captor and XVCL approaches also presented high values for the CK scattering metric. The product derivation assets from these approaches are composed by a set of specific files that describe transformation rules, which represent a significant part of the CK specification. Nonetheless, code assets also contain CK specification in the form of tags. This explains why these two approaches got quite similar values for CK scattering metrics in all MobileMedia releases. As we see in Figure 1, the collected values for the CK scattering metric were higher for the Captor and XVCL approaches.

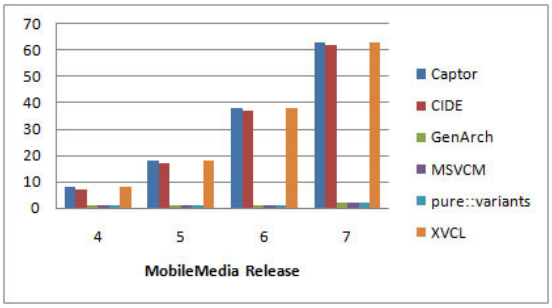

Figure 1. Configuration Knowledge Scattering.

Figure 7 illustrates the product derivation artifacts specified for five approaches that are responsible for the inclusion of the Photo and Music features in the MobileMedia SPL. For example, XVCL requires that each SPL asset that is processed to be a frame. In order to transform an asset in a frame, we have to include basic tags with some parameters, whether this frame has fine-grained variability or not. In the right side of Figure 7 for example, PhotoAndMusicAspect.aj.xvcl represents the correspondent aspect specified as a frame. On the other hand, Captor also needs to use XSL tags, and every variable asset has to be a template. The right side of Figure 7 also shows PhotoAndMusicAspect.aj aspect specified with a set of XSL tags used by Captor derivation artifacts. The left side of Figure 7 illustrates the configuration files of the Captor and XVCL approaches that are used to specify the transformation rules mentioned above. The scattering of these transformation rules and code tags along product derivation artifacts and code was quantified by the degree of scattering metric. Note that GenArch, MSVCM and pure::variants does not contain CK information on the code asset. The CK is modularized into a single file.

Configuration Knowledge Tangling inside Code Assets. Figure 2 shows the collected values for CK tangling metric. The Captor and XVCL approaches presented higher values for the tangling metric. This mainly occurs because both approaches must contain specific headers and footers in any code asset associated with CK. This information is used during the product derivation process of each approach. Figure 7(right side) shows, for example, that the PhotoAndMusicAspect.aj aspect needs to include configuration tags for both Captor and XVCL approaches. All the code assets associated with variabilities required the insertion of these same configuration tags thus contributing to improve the CK scattering. Precisely because GenArch, MSVCM and pure::variants, have the CK properly insulated, there is no CK tangling inside code asset.

The CK tangling metric was quantified in CIDE by the occurrence of color sentences spread along code assets. Because of that, CIDE initially presented a low tangling considering the MobileMedia release 4, but as the number of features increased, values got higher compared to other approaches. It mainly

happens because with the increase of features to be managed, feature expressions also become more complex. GenArch, pure::variants and MSVCM had equivalent and the best results for the CK tangling metric considering all the releases. On the first three releases investigated (releases 4 to 6), the CK tangling for these approaches was zero. This occurred because the CK was adequately modularized into a dedicated model.

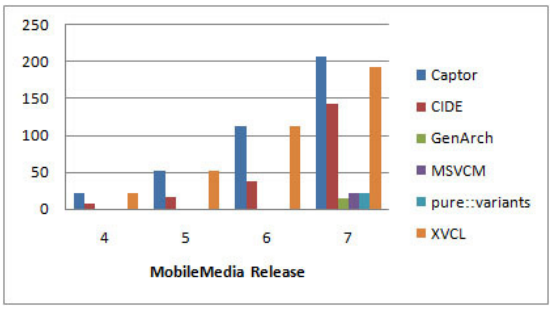

Figure 2. CK Tangling inside Code Assets.

We can also observe that fine-grained variability increases CK tangling in code assets. The fact that most of MobileMedia variations are well modularized with aspects contributed for the lower values of the scattering and tangling metrics in releases 4-6. However, release 7 has a particular case where one asset (OptionalFeatureAspect.aj) contains fine-grained variability that cannot be handled using only the CK model. Figure 3 illustrates this aspect, where the arguments passed on the declare precedence clause are variabilities and depend on the selection of specific features to be included in the aspect code. Because of that, this file needs to be processed using conditional compilation tags (MSVCM and pure::variants) or template processing (GenArch and Captor). Both techniques are used during product derivation to decide if part of this asset will be included or not. This kind of fine-grained variability usually happens on legacy SPLs[14]. Note that all the approaches and metrics from this study suffer influence of this fine-grained variability. This fine-grained variability was responsible for the light increase in the collected results for the CK tangling metric considering the Captor and XVCL for the release 7 of MobileMedia.

```
01 package lancs.mobilemedia.optional;
   ...
20 public aspect OptionalFeatureAspect {

21  declare precedence : SMSAspect,
    CopyMultiMediaAspect, FavouritesAspect,
    SortingAspect,  PersisteFavoritesAspect;

22 }
```

Figure 3. OptionalFeatureAspect.

3.2 Complexity Analysis

The main goal of the two metrics on this group is to measure the complexity/size of the CK specification in each approach, and furthermore analyze how they behave when evolving the SPL.

Number of tokens in CK specification. The number of tokens metric allowed us to distinguish the size and complexity of CK specification on the different derivation approaches. Figure 4 shows the collected values for this metric. Captor is the approach that requires the higher number of tokens in CK specification. Reasons for that include the need for creating several CK decision expressions (task calls and definitions), and headers and footers specified in the code templates. XVCL also presented higher values for the number of tokens in CK metric. This happens due to the same reason of Captor, except that task definition and call is specified directly in XVCL files. In MSVCM, values for this metric are higher than GenArch since it is necessary to specify asset names and mandatory associations as well.

Figure 7 shows how we count tokens in all approaches (except CIDE), for the case of the PhotoAndMusicAspect asset. This aspect is related to the joint selection of the Photo and Music features. For this metric, in XVCL, Captor, GenArch, MSVCM and pure::variants approaches, we count 94, 129, 3,3 and 15, respectively. In the CIDE approach, since we just color code elements with associated features, there is no textual CK specification. Therefore, the metric values for all releases are zero. GenArch, pure::variants and MSVCM have values much smaller than other approaches. In GenArch, the architecture model that abstracts all the code assets is built automatically by the tool. Product line engineers only need to write feature expressions, associating features to assets. So, metric values tend to be lower for GenArch. This happens similarly in MSVCM.

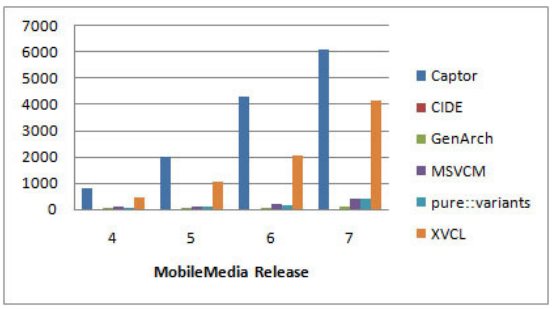

Figure 4. Number of Tokens in CK specification.

```
<select option="Photo">
 <option value="yes">
  <adapt x-frame="PhotoAspect.aj.xvcl"  outfile="PhotoAspect.aj" />
  <adapt x-frame="ImageAlbumData.java.xvcl" outfile="ImageAlbumData.java"/>
  <adapt x-frame="Image...java.xvcl"    outfile="ImageMediaAccessor.java" />
  <adapt x-frame="PhotoViewScreen.java.xvcl"outfile="PhotoViewScreen.java"/>
  <adapt x-frame="SCreensAspectEH.aj.xvcl" outfile="SCreensAspectEH.aj" />

<select option="Photo">          Release 06->07
 <option value="yes">
  <adapt x-frame="PhotoAspect.aj.xvcl"  outfile="PhotoAspect.aj" />
  <adapt x-frame="ImageAlbumData.java.xvcl" outfile="ImageAlbumData.java"/>
  <adapt x-frame="ImageMedia.java.xvcl" outfile="ImageMediaAccessor.java" />
  <adapt x-frame="PhotoViewScreen.java.xvcl"outfile="PhotoViewScreen.java"/>
  <adapt x-frame="SCreensAspectEH.aj.xvcl" outfile="SCreensAspectEH.aj" />
  <adapt x-frame="PhotoSelector.aj.xvcl" outfile="PhotoSelector.aj" />
  <adapt x-frame="AbstractPhoto.aj.xvcl"outfile="AbstractPhotoAspect.aj" />
  <adapt frame="PhotoNot...aj.xvcl" outfile="PhotoNotVideoNotMusic.aj" />
```

Figure 5. XVCL – Sentence Expression.

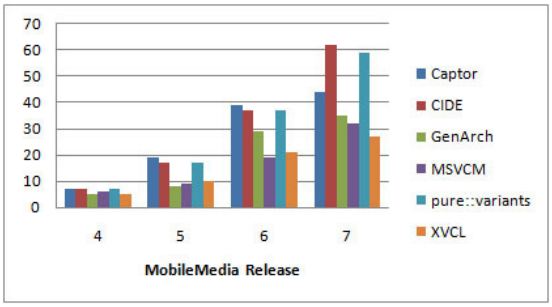

Figure 6. Sentence Expressions in CK Specification.

Sentence Expressions in CK specification. Figure 6 shows the number of sentence expressions for the different MobileMedia

Figure 7. Approaches and Metrics.

releases specified in the six approaches. Similarly to the tangling metric, the CIDE approach presented a considerable increase when adding new features. A sentence expression in CIDE is considered as a colored element. Therefore, these metric values are somewhat correspondent to the scattering metric values for the CIDE approach. The XVCL and MSVCM approaches can group many assets into a single sentence. A difference is that in MSVCM, we also specify the mandatory relationships as previously mentioned. Figure 6 illustrates this grouping characteristic of XVCL. It shows that, in release 6, the Photo feature is associated with 5 assets. In the next release, 3 new assets are added to this expression, and no other sentence needs to be created. In Captor, pure::variants and GenArch, we need to add 3 sentence expressions. This can be mitigated in pure::variants and GenArch by associating higher level abstractions, such as packages, with feature expressions. However, if feature implementation is highly scattered, it might not be possible to do so. In Figure 7 the sentence expressions are showed for the different approaches.

3.3 Stability Analysis

This analysis looks at three different perspectives of sentence expressions between releases: added, modified and removed.

Added. Figure 8 (a) shows the number of sentence expressions added in the configuration knowledge of each approach between releases. When adding code assets, GenArch, pure::variants, Captor and CIDE treat assets individually, so they have larger effort than MSVCM and XVCL, which can group these assets. In GenArch and Captor, the inclusion of an asset represents a new sentence expression added to the configuration knowledge.

Changed. Figure 8(b) shows the number of sentence expressions changed in the configuration knowledge. The releases 4, 5 and 6

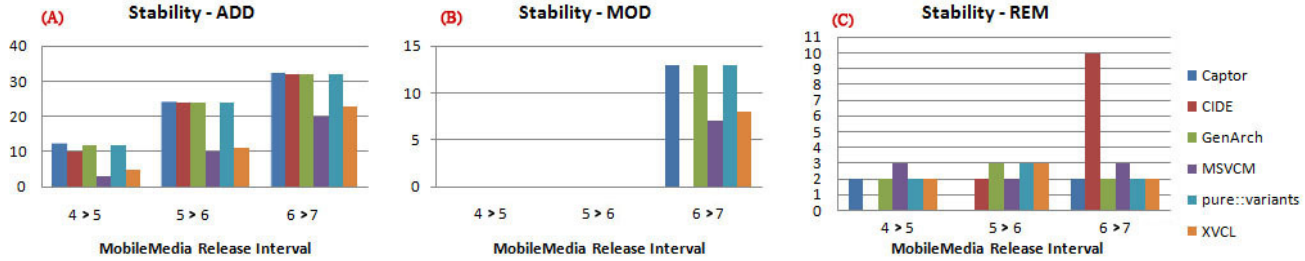

Figure 8. Stability – Added (A), Modified (B) and Removed (C) .

had many new assets and few expressions, or none in most cases, that could be reused. Again, the approaches (GenArch, Captor and pure::variants) that treat assets individually had larger effort than the other ones. The CIDE approach has not changed items, because when changes occur in an asset, you must remove its color and then mark the new artifact with this color.

Removed. Figure 8(c) shows the numbers of sentence expressions removed from the CK in each approach. This usually occurs when code assets are removed. These metric results are very similar to almost all approaches, variation happens only in cases that the approach could reuse some expressions, reflecting on Figure 8(b). The fact that CIDE cannot change the sentence expressions results in the increase of the number of removed items.

4. DISCUSSIONS

In this section, we discuss interesting issues and lessons learned related to the assessment of the product derivation approaches in our preliminary evolution study.

General Analysis of the Study Results. The GenArch, pure::variants and MSVCM approaches presented the best results considering most of the modularity and complexity metrics. The main reason that contributed to the good performance of these approaches is the use of a separated CK specification. Captor and XVCL obtained the less satisfactory results considering the modularity and complexity metrics. This mainly occurred due to the textual and complex nature of CK specification provided by these approaches. Considering initial MobileMedia releases, CIDE presented reasonable results for the modularity and complexity metrics. However, it did not scale along the subsequent ones in terms of CK scattering and tangling, because of the increasing number of CK color sentences specified in the code assets. Regarding the stability metrics, we have observed that Captor, GenArch, MSVCM and XVCL required to add and modify a small and stable number of CK sentence expressions along the different releases. MSVCM was the approach that presented the better stability for the CK expressions. On the other hand, CIDE exhibited higher values for the number of removed CK expressions due to the need to reassign new colors to CK sentences that are changed in the code assets along the releases.

Automated and Model-driven Tools. Our study has revealed that the CK specification in a dedicated model or file can bring several benefits to the modularization, complexity and stability attributes. That was the case for GenArch, MSVCM and pure::variants approaches. Captor and XVCL did not obtained better results in most of the modularization and complexity metrics. These approaches require manually written transformation rules related to the product derivation process. It is interesting to notice that the simple automated support to the specification and generation of the transformation rules that represents CK in both Captor and XVCL approaches could bring equivalent results to the other approaches.

CK Specification inside Code Assets. Another interesting issue that our study has also revealed is the positive/negative impact of the variability implementation technique in the modularization of the product derivation artifacts. Most of the variabilities implemented for MobileMedia releases were codified using aspect-oriented programming. This contributed to the positive values obtained for most of the modularity, complexity and stability for the different approaches. Analysis of the MobileMedia implementation using conditional compilation (CC), for example, would find a large number of CC directives (IFDEFs) spread and tangled along the different code assets. All these CC directives can also contribute to the instability of the product derivation artifacts during SPL evolution because they are not adequately modularized. Although the adoption of aspect-oriented programming in the MobileMedia implementation brought advantages and benefits to the specification of the product derivation artifacts, our study also illustrated that fine-grained variabilities encountered inside code assets can lead to difficult CK modularization scenarios, even for approaches that provides a separate CK specification. This was illustrated in our study by the OptionalFeatureAspect that establishes the precedence between aspects (variabilities) that will be applied to the SPL core.

5. RELATED WORK

To the best of our knowledge, we are not aware of empirical research on the assessment of different product derivation tools used in the evolution context for SPLs. Many works evaluate a single approach for SPL implementation used in a SPL evolution scenario. The original MobileMedia study [12] compares the negative and positive impact of using aspects for implementing SPLs, instead of conditional compilation. There are some similarities to our work, since we are using the same releases and adapting some of the used metrics. So, instead of evaluating the code only, in this work we focus on how different CK representations provided by the product derivation tools behave in a SPL evolution scenario.

Mannisto et al. discuss the problem of managing configuration knowledge evolution [17], describing key elements and presenting an outline for future work. They present an approach based on the Generic Product Structure Model and directions on how this approach should be used. However, they do not evaluate its use in a case study, as we focus on this work.

Dhungana et al. present a tool-supported approach for treating evolution in Model-based Product Line Engineering [11]. The approach decomposes large variability models into sets of interrelated model fragments. Such fragments are merged to provide full variability models. They report an experience of using it in an industrial collaboration. Some of the approaches we evaluate in this work (GenArch and pure::variants) are model-based and provide similar functionality. A difference to our evaluation is that due to model decomposition, they evaluate multi-team support for creating and maintaining SPL variability models. This is not something we investigated in this work, but it could be targeted in future work.

Many works compare product derivation tools in a systematic, but general way, not specifically focusing the CK. Rabiser et al. aim to identify and validate requirements for tool-supported product derivation [18]. Requirements are identified in a high-level way through a systematic literature review and validated through an expert survey. Sinnema and Delstra classify six variability modeling techniques [21] using a running example. They define a classification framework that lists a number of aspects that a variability modeling technique should possess. Lisboa et al. present a systematic review of domain analysis tools [16]. They

focus on identifying (i) whether current available tools support a specific or generic process; (ii) their main functionalities; and (iii) the development context, and where it is being used --- academia or industry. Functionalities identified are categorized in priority levels: *essential*, *important*, and *low*. Khurum and Gorschek also conducted a systematic review of domain analysis solutions, but only for tools that focus on software product lines [15]. They focus on the usability and usefulness of the existing solutions, in terms of scalability of introduction and use, better alternative investment, and effectiveness. These works focus on comparing functionalities that existing product derivation tools provide. As mentioned, they do not focus on the configuration knowledge, nor evaluate tools in the evolution context. Khurum and Gorschek findings [15] indicate an absence of qualitative and quantitative results from empirical application of the approaches. Such absence, they argue, complicates the task of evaluating usability and usefulness of existing solutions. In this work, we attempt to fill this gap, through a quantitative and qualitative assessment of the configuration knowledge using different releases of the MobileMedia SPL.

6. CONCLUSIONS AND FUTURE WORK

This paper presented a preliminary empirical study that focused on comparing different product derivation approaches and their configuration knowledge representation. We performed this comparison through a quantitative analysis that measured modularity, complexity and stability attributes along product derivation artifacts produced for four evolution releases of a mobile SPL. Different metrics were used to quantify these attributes in terms of scattering, tangling, number of tokens and amount of CK sentence expressions, and number of added/removed/changed CK expressions considering all the SPL releases. We noticed that the adoption of specific approach strategies can lead to positive results in terms of the investigated attributes. From a general analysis, we concluded that approaches with a dedicated model or file to represent the CK specification had better results on many aspects. It was also observed that modest adaptations in some existing tools and approaches can bring a significant improvement on their performance, such as: (i) the automatic generation of headers and footers on Captor and XVCL tools; and (ii) the introduction of the capability to relate different sentence expression to two or more assets.

As a future work, we plan to replicate our study for different SPL domains and different variability implementation techniques in order to observe if the same obtained results of this study can be found in other SPL evolution scenarios. Besides, we also intend to include other existing product derivation tools in our future empirical studies. We intend to define and run more controlled experiments and case studies that follow the guidelines of the empirical software engineering. Last but not least, we are also working to adopt many of the findings and guidelines provided by our study in the design and implementation of our product derivation tools.

Acknowledgements. This work is supported in part by Brazilian Council on Research (CNPq), grants 313064/2009-1 and 480978/2008-5 ; and CAPES/PROCAD, grant 090/2007.

7. REFERENCES

[1] J. Barreiros and A. Moreira. A Model-based Representation of Configuration Knowledge. In FOSD '09:, pages 43–48, New York, NY, USA, 2009. ACM.

[2] P. Bassett. Framing Software Reuse - Lessons from Real World. Prentice Hall, 1997.

[3] D. Beuche. Modeling and Building Software Product Lines with pure::variants. In SPLC, page 358. IEEE Computer, 2008.

[4] R. Bonifácio and P. Borba. Modeling Scenario Variability as Crosscutting Mechanisms. In AOSD'09, pages 125–136, USA.

[5] R. Bonifácio, L. Teixeira, and P. Borba. Hephaestus: A Tool for Managing Product Line Variabilities. In III SBCARS 2009 – Tools Session, pages 26–34, Natal, RN, Brazil, 2009.

[6] E. Cirilo, et al. Integrating Component and Product Lines Technologies. In H. Mei, editor, ICSR, volume 5030 of Lecture Notes in Computer Science, pages 130–141. Springer, 2008.

[7] E. Cirilo, et al. A Product Derivation Tool Based on Model-Driven Techniques and Annotations. J. UCS, 14(8):1344–1367, 2008.

[8] P. Clements and L. M. Northrop. Software Product Lines: Practices and Patterns. Professional. Addison-Wesley, 2001.

[9] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3):621–645, 2006.

[10] S. Deelstra, et al. Product derivation in software product families: a case study. Journal of Systems and Software, 74(2):173–194, Jan. 2005.

[11] D. Dhungana, T. Neumayer, P. Grunbacher, and R. Rabiser. Supporting evolution in model-based product line engineering. Proceedings of SPLC '08, IEEE Computer Society.

[12] E. Figueiredo, et al. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. Proceedings of of ICSE'08, pages 261–270, New York, NY, USA, 2008. ACM.

[13] P. Junior and C. A. de Freitas. Geração de aplicações para linhas de produtos orientadas a aspectos com apoio da ferramenta Captor-AO, *MSc Dissertation*, University of São Paulo, Nov. 2008.

[14] C. Kastner, S. Apel, and M. Kuhlemann. Granularity in Software Product Lines. Proceedings of ICSE 2008, Leipzig, Germany, May 10-18, 2008, pages 311–320. ACM, 2008.

[15] M. Khurum and T. Gorschek. A Systematic Review of Domain Analysis Solutions for Product Lines. Journal of Systems and Software, 82(12):1982 – 2003, 2009.

[16] L. B. Lisboa, et al. A Systematic Review of Domain Analysis Tools. Information and Software Technology, 52(1):1 – 13, 2010.

[17] T. Mannisto, H. Peltonen, and R. Sulonen. View to product configuration knowledge modelling and evolution, Oct. 21 1996.

[18] R. Rabiser, et al. Requirements for Product Derivation Support: Results from a Systematic Literature Review and an Expert Survey. Information and Software Technology, 52(3):324 – 346, 2010.

[19] A. Garcia, et al. Modularizing Design Patterns With Aspects: A Quantitative Study. Proceedings of AOSD´2005, pp. 3-14, Chicago, 2005.

[20] E. K. Shimabukuro Junior. Um Gerador de Aplicações Configurável, *MSc Dissertation*, University of São Paulo, 2006.

[21] M. Sinnema, S. Deelstra. Classifying Variability Modeling Techniques. Information and Software Technology, 49(7):717 – 739, 2007.

[22] S. Swe, et al. XVCL: A Tutorial. In Proceedings of SEKE´2002, Ischia, Italy, 2002.

[23] S. S. Yau and J. S. Collofello. Design stability measures for software maintenance. IEEE Transactions on Software Engineering, 11(9):849–856, Sept. 1985.