A Comparative Study of Compositional and Annotative Modelling Approaches for Software Process Lines

Fellipe A. Aleixo[1,2], Marília Freire[1,2], Daniel Alencar[1], Edmilson Campos[1,2], Uirá Kulesza[1]
[1]Departamento de Informática e Matemática Aplicada – PPgSC/CCET/UFRN , Natal-RN, Brazil
[2]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte – IFRN, Natal-RN, Brazil
{fellipe.aleixo, marilia.freire, edmilson.campos}@ifrn.edu.br, daniel.calencar@gmail.com, uira@dimap.ufrn.br

***Abstract*—This paper presents a comparative study of compositional and annotative modeling approaches for software process lines. In our comparative study, OpenUP and Scrum based software process lines extracted from existing projects are modeled and implemented using an existing compositional approach – EPF Composer, and an annotative approach – GenArch-P, with the main aim to address a systematic variability management and automatic process derivation. In order to promote the analysis of the approaches were adapted some comparison criteria previously adopted in other studies. Our study results show that the annotative approach can bring many advantages to the modeling of software process lines considering our comparison criteria. On the other hand, our study also concludes that many existing compositional mechanisms should also be integrated with annotative approaches in order to improve the modularity of process elements associated to specific kinds of process variabilities.**

Software Process; Variability Management; and Software Process Lines

I. INTRODUCTION

The application of the concepts and tools of software product line (SPL) engineering [1] to the context of software process is called software process line engineering. The main objective of software process lines is to promote the large-scale reuse in software process families. To achieve this goal, software process line allows the automatic customization of software process to specific enterprise contexts and scenarios. Over the last years, new approaches and techniques were developed to address the modeling of software process lines. The evolution of the research in the software product line engineering is often reflected to the software process lines.

The research work about software process lines has grown in recent years and has reached important results. The following topics have been explored: (i) definition and motivation for software process line engineering [2] [3] [4]; (ii) visual representation of process variabilities [5][6]; (iii) scoping of software process lines [7]; and (iv) modeling of software process lines variabilities [8] [9]. While some initial research questions have been answered, new ones have emerged and need to be addressed. Examples of existing challenges related with the software process line engineering are: what are the available approaches for the modeling of software process lines? Which techniques and tools should be used in each specific scenario? Which are the benefits and disadvantages of each of these approaches?

This work presents a comparative qualitative study of compositional and annotative approaches for modeling software process lines. The compositional approach is represented by the EPF Composer [10], an industrial software process engineering tool, that supports the modularization and composition of process elements and the selection of those elements to generate a customized process specification. The annotative approach is represented by GenArch-P (GenArch for Process) [9], an adaptation of an existing model-based product derivation tool [11], which provides support to create generative models that express the process variabilities and associate them with existing process elements.

In order to conduct our comparative study, two software process lines were specified and modeled: (i) the first one was based on an Open-UP process family – extracted from three existing projects; and (ii) the other one was based on a Scrum process family. Both process lines were then modelled and implemented using EPF Composer and GenArch-P with the main aim to address a systematic variability management and the automatic process derivation. After that, we compare the model elements and composition mechanisms of the different approaches using a comparison criteria proposed in a previous work in the context of software product line [12][13][14]. Our study also discusses the integration between existing compositional and annotative techniques, in order to improve the variability management in software process lines.

The remainder of this paper is organized as follows. Section II provides an overview of the two investigated model-driven approaches. Section III describes our study settings with the target software process lines and the comparison criteria. Section IV presents the modeling of the software process lines and the obtained results. Finally, Section V presents the final conclusions and directions for future work.

II. Model-driven Approaches for Software Process Lines

Over the last years several approaches have been proposed to the development of software process lines. However there are two of them that stand out from the others because they focus on the modeling and automatic derivation of process variabilities. One of these approaches is EPF Composer [10] that allows modularizing process elements using compositional refinement techniques. The other one is called GenArch-P [15] [9] that promotes the variability management of process artifacts using annotation-based techniques. Next sections will cover these two approaches in more details.

A. Compositional Approach: EPF Composer

The EPF Composer is a process authoring tool that is part of a software process engineering framework that allows the authoring, tailoring and deployment of software processes. In EPF the process contents are organized as method contents and workflows. These method contents [10] describe what is going to be produced, the necessary skills, the explanations about how a task has to be made, and so on. Process variabilities in EPF can be handled with specific refinement mechanisms, called method content variabilities, which are: *extends*, *contributes*, *replace*, and *extends and replace* [10].

Using the *contributes* variability mechanism, a specialized process element can append contents to attributes of other process element (base element) in a way that it does not replace the base content. For example, if the contributing element defines an additional description to an attribute, this description is appended at the end of the description of the same attribute of the base element. When the process is published the base element appears with the new contributions. The *replace* variability mechanism allows one base element to be replaced by a new element. The replace variability provides a way to an element replace another one without modifying its properties. When the process is published, the replacing element is presented, while the base element is left untouched and is not shown. Adopting the *extends* variability mechanism, a specialized element inherits the attributes from a base element and can override these attributes. When the process is published, both elements (the base and the specialized) are published in the process. Finally, the *extends and replace* variability mechanism combines the effects of extends and replaces variabilities into one variability strategy. While the replace variability replaces all contents from the base element, this kind of variability acts as a "selective replacement" where the replaced contents are just those that were redefined.

B. Annotative Approach: GenArch-P

GenArch-P is a tool for software process derivation adapted from a model-driven derivation approach for software product lines, called GenArch [11][15]. The tool works with an annotative approach, where the variable elements of a software process can be annotated and associated to existing features. The annotative approach is widely used in various software product lines tools, such as pure::variants, Gears and CIDE.

GenArch-P tool specifies two models: (i) the process model, and (ii) the feature model. The process model is generated from the parsing of the process specification model. After that, the elements of the process model can be annotated to reflect their association with optional and/or alternative features of the software process line. The annotations in GenArch-P are based on the underlying structure of the process model where each element can have an associated variation property that describes the element variation type and the feature parent. After the annotation of variabilities in the elements of the process model, this model specifies the configuration knowledge. The configuration knowledge defines the mapping between features and their associated process elements.

After the annotation of variabilities in the process model, GenArch-P automatically generates the feature model. The feature model can be used to model and specify the SPL commonalities and variabilities. The model presents all the possible options to be selected in terms of features. To derive a new process, the user creates a new feature configuration and selects the desired features for the new process. After that, the GenArch-P is ready to automatically derive the process instance according with the selected features.

III. Study Settings

In order to allow the comparative analysis of compositional and annotative approaches we have modelled two software process lines with different kinds of process variabilities using the evaluated modelling approaches. One of the modeled process lines was based on OpenUP process family, and the other one was based on Scrum process family. After the specification of the process line using EPF and GenArch-P, we have conducted a comparative analysis based on modularity criteria adapted from a previous study [12] [13]. Subsection III.A describes the study phases and discusses the assessment procedures. Subsection III.B gives an overview of the software process lines used in our study. Subsection III.C describes the comparison criteria adopted in our study to address the modularity analysis of the investigated approaches.

A. Study Phases and Assessment Procedures

The study was organized in five major phases, which are: (1) definition of the scope of the chosen software process lines; (2) the variability identification of the software process lines; (3) the modeling of the software process lines using the EPF Composer and GenArch-P approaches; (4) the derivation of processes instances; and (5) the analysis of the variability management mechanisms of the investigated approaches and assessment of the results based on the defined criteria. Next, we provide details of each study phase. The study was conducted by five researchers: one PhD researcher, two PhD students and two MSc students. .

The first phase was responsible for the selection of real processes from existing projects, and analysis of documentation from these projects to be used as basis for commonalities and variabilities identification. This analysis was then used for the definition of the scope of the software process lines. The second phase was responsible for modelling the software process lines with the definition of their common and variable process elements.

The modeling of the software process lines using the investigated approaches happened in the third phase. In the

fourth phase of the study, we have automatically derived software processes instances from the specified process lines, where each process instance was defined as a selection of a set of variable features. This step was useful for assessing and validating the modeled software process lines.

The fifth and final phase was responsible to summarize the results of the study, and analyze these results from the perspective of the defined criteria. The criteria assessment was chosen to allow analyzing the modularity and variability management of the investigated approaches.

B. Target Software Process Lines

The first phase in our study was the definition of the software process lines that should be used in the comparison of the investigated approaches. It was defined two software process lines: the first one was extracted from OpenUP customizations and the second one was extracted from Scrum customizations.

The scoping of the OpenUP process line was defined through the analysis of three OpenUP-based processes from existing research and development projects developed in cooperation between our institutions and the industry. The first project was the development of a software system for auditing telephony networks. This system performs the counting, summarization, and the analyses of the connection records created by a specific hardware. The second project involved the development of a module of a distributed system responsible for the collection and the storage of the information related to the federal institutions of professional and technological education in Brazil. The third, and last, project comprised the implementation of an integrated academic and administrative management system for the federal institutions of professional and technological education in Brazil.

The scoping of the Scrum process family was established based on our industry experience and the investigation of several instantiations of the Scrum process published in the Internet, such as: (i) Scrum.org (www.scrum.org); (ii) Scrum Alliance (scrumalliance.org); (iii) Scrum from Mountain Goat Software (mountaingoatsoftware.com/topics/scrum); (iv) Scrum from EPF [10]; and (v) Scrum from CodeProject (www.codeproject.com/Articles/4798/What-is-SCRUM)..

The complete specification of the process lines with the selected approaches was accomplished using the extractive technique. It comprised the analysis of the commonalities and variabilities of the existing software process families. Each process line specifies: (i) the core of the process line – that aggregates its common process elements; and (ii) the process line variabilities – that represent variable elements from the investigated processes that are modelled separately and associated to high-level features.

Table I shows examples of high-level features and their related variable process elements for the OpenUP based process line. We have specified about 75 process elements as part of the core of this process line. In addition, it was found 9 optional features, 9 alternative features, and 4 OR-features. Table II shows the names of the identified features and the number of associated process elements. The identified features are a direct result of the analysis and extraction of commonalities and variabilities of the investigated processes.

TABLE I. EXAMPLE OF HIGH-LEVEL OPENUP PROCESS FEATURES AND THEIR RELATED SOFTWARE PROCESS ELEMENTS

Feature Example	Some Associated Process Elements
Use case specification technique (alternative)	**(1)** Role: Analyst **(2)** Technical practice: Use Case Driven Development **(3)** Work product: Use-Case Model **(4)** Work product: Use Case **(5)** Checklist: Use Case **(6)** Checklist: Use Case Model **(7)** Concept: Actor **(8)** Concept: Use Case
Additional elements from the Scrum Process	**(1)** Work product: Work Items List **(2)** Guidance: Risk List **(3)** Guidance: Work Items List **(4)** Concept: Retrospective **(5)** Concept: Risk **(6)** Example: Iteration Burn Down Report **(7)** Example: Project Burn Down Report

TABLE II. FEATURES FOR THE OPENUP PROCESS LINE

Feature	Alternatives (if there is)	Related process elements
Complementary Process Practices		
Additional elements from the Scrum Process		25
Requirements techniques and technologies		
Specification techniques	Use cases	27
	Users stories	27
	Product backlog	27
Specification tools	Asta Community	24
	Rational Software Architect	24
	Borland Together	24
	ArgoUML	24
Design techniques and technologies		
Architecture documentation	Agile design	64
	Well documented architecture	16
Implementation techniques and technologies		
Used language (OR-feature)	Java	7
	C#	7
	Ruby	7
	Phyton	7
Use of JEE framework		4
Use of Eclipse IDE		5
Use of JUnit framework		15
Continuous integration techniques and technologies		
Use of Hudson tool		8
Metrics techniques and technologies		
Use of Maven tool (code metrics mined from SVN)		5
Metric for assessing the activities progress		3
Metric for assessing the deadlines fulfillment		3
Metric for assessing the duration of main activities		3

During our analysis, we have also found constraints between features from the process line. For example, the features representing the usage of JEE and JUnit frameworks required the selection of Java feature as the programming language. Another constraint example was found at the selection of the architecture documentation according with the agile design, which disabled the feature selection of the design specification tool, since in agile development, the software design is usually made in paper and whiteboards [16].

Table III presents the features found for the Scrum based process line. Differently of the OpenUP process line, it was not clearly found high-level features. Most of the variabilities occurred only in process elements. The features of the Scrum process line address a wide range of customizations, from the more traditional ones to the more minimalists. Another aspect of the Scrum process line is that some elements presented some variations inside of it, such as: (i) the sprint planning meeting can optionally include the planning poker practice; (ii) the product backlog can alternatively be prioritized by complexity or by business value; (iii) the product backlog item is usually defined as a code module, but it can be of other type as: implementation phase plan, implementation report, detailed design, user interface definition or unit test.

TABLE III. MAIN FEATURES FOR THE SCRUM BASED PROCESS LINE

Scrum Process Element	Type of Element	Type of Feature
Sprint planning meeting	Ceremony	Mandatory
Daily meeting	Ceremony	Optional
Sprint review meeting	Ceremony	Optional
Retrospective	Ceremony	Optional
Product backlog	Artifact	Mandatory
Product backlog item	(Artifact) Concept	Mandatory
Sprint backlog	Artifact	Mandatory
Burn down charts	Artifact	Optional
Impediment registry	Artifact	Optional
Scrum board	Artifact	Optional
Release plan	Artifact	Optional
Story cards	Artifact	Optional
Scrum master	Role	Mandatory
Scrum team	Role	Mandatory
Product owner	Role	Optional
Velocity	Metric	Optional

C. Comparison Criteria

In order to promote the assessment of software product line implementation techniques, Kästner et al [12] have defined comparison criteria for such purpose. In our work, we have adapted six of these criteria – *modularity*, *traceability*, *error detection*, *granularity*, *uniformity*, and *adoption* – to the context of software process lines, which are presented next. In addition, we have also analyzed the approaches support for systematic variability management. This work introduces a new criterion named *systematic variability management*. Next these criteria will be briefly explained.

The **modularity** criterion allows analyzing the modularization of the feature implementation using the different approaches. An adequate modularization allows isolating the implementation of a specific feature and reducing the overall complexity of the code [12]. With the appropriate feature isolation, the developer can understand, check and test them independently, thus contributing to lower development and maintenance costs. In our work, this criterion has the same purpose, with the only difference that the isolation is analyzed for the process elements associated with specific features.

The **traceability** criterion analyzes how easy is the visualization of the mapping between features and process elements. In Kästner [12], the visualization of the assets associated with a given feature allows the developer look up directly the corresponding code for enhancing or fixing a bug in the feature. This visualization is also important to process lines to promote the trace of existing process elements associated with specific high-level features.

The **error detection** criterion is intended to analyze how existing approaches provide support to consistency checking of the software process line and their derived processes. In Kästner [12], the error detection is useful to avoid the expensive maintenance late in development cycle to correct type, syntactic and semantic errors. In the context of our study, this criterion is related with the consistency checking of the specified elements of the process line. Examples of such checking are a reference for a non-present element or an inconsistence in the flow of process activities.

The **granularity** criterion refers to the approach support to associate features with process elements of different granularity. A software process line can also be composed of coarse and fine-grained variabilities. Because of that it is fundamental that process line modeling approaches provide support and different mechanisms to model process variabilities from different granularity levels.

Kästner et al [12] defines the **uniformity** criterion as the ability to provide uniform support for multiple (code and non-code) languages. In the software process line context, this criterion is responsible to evaluate how a particular approach is attached to a specific software process specification. Hence, the possibility of working with software processes specified using different meta-models is analyzed for each approach. The fact that the approach is restricted to a single meta-model for specifying the software processes makes the approach less generic, and discards the advantages of other forms of specification.

The **adoption** criterion analyses the difficulty of adopting existing approaches in terms of the amount of required knowledge for the approach application. Similar to the Kästner [12] work, we investigate the amount of concepts, mechanisms and tools needed during the application of the approach.

In **systematic variability management**, it is analyzed the approach mechanisms for systematic and effective variability management. Two aspects are analyzed in this criterion: (i) the approach mechanisms for specifying the variabilities, their respective constraints, and the mapping with process elements; and (ii) the approach support to automated process derivation from existing process assets.

IV. STUDY RESULTS

In this section, we report our study results by describing the process line definition using EPF Composer (Subsection IV.A) and using GenArch-P (Subsection IV.B). In addition, we also present and discuss the obtained results of the two approaches in the comparison criteria (Subsection IV.C). Finally, some open issues are presented and discussed (Subsection IV.D).

A. Process Line Modelling using EPF Composer

Process line engineering. The definition of the process lines was accomplished taken as basis the originals EPF method plugins [10] for OpenUP and Scrum. The first step of the process line definition was the creation of a new method plugin for each process line. Next, it was created the mandatory process elements. These mandatory elements were organized

inside a content package, named "core". In order to address the definition of the process elements related to optional and alternative features, we have created specific content packages to each feature. To facilitate the organization, the names of the features were used in the related content packages.

Figure 1 presents the set of content packages created to define the OpenUP based process line. The names of the content packages indicate that the correspondent feature is optional or alternative, and also identify the feature. The definition of the Scrum based process line was different due to two factors, which are: (i) it was not identified high-level features, and (ii) the Scrum process does not imposes an activities flow. The variability analysis found optional and alternative process elements. Specific content packages were created to accommodate the variability of a single process element. Figure 2 presents the content packages created to group the process elements associated to specific features in the Scrum based process line. In each content package, besides the variable elements, it is also created specializations of the related mandatory process elements.

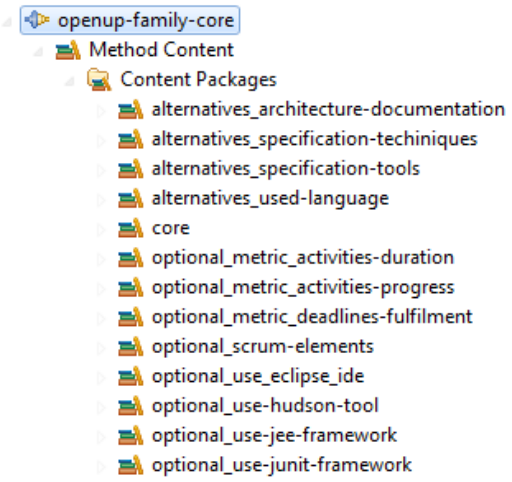

Figure 1. Content packages created for OpenUP software process line

The workflow of the software process lines was specified using the capability patterns structure provided by the EPF. We have defined the workflow of the core of software process lines referencing the mandatory process elements discussed previously. One capability pattern, named “core”, was used to specify the flow of activities aggregating all the mandatory process elements. After that, we have created some capability patterns to encapsulate the increments to the core capability pattern in order to address all the features that demand increments in the core activities flow. The names of the correspondent features were used to entitle the created capability patterns. Because the Scrum based process line does not define an activity workflow, it was not necessary the usage of capability patterns in this process line. The execution of a Scrum process is based on work cycles, called sprints, transforming the requirements of the software product (product backlog) in partial increments in product.

The fine-grained variabilities that happen inside the Scrum process elements were modeled using EPF with the inclusion of specialized elements that contribute, extend or replace the content of the mandatory process elements or just interacts with some core process elements. For example, the increment of a description of a new specific type work item in the product backlog artifact is accomplished specializing the product backlog with *contributes* variability mechanism. The specialized element appends the main description of the product backlog, describing the new type of work item, and it also appends the necessary steps of the task to include the instructions to collect items of this specific type.

Figure 3 shows the set of capability patterns created for the OpenUP process line. Each of these capability patterns uses contributes variability mechanism to refine the core capability pattern. The *contributes* variability mechanism is the only used in this case, because the content of the core have to be maintained and the feature content has to be added to it. Table IV presents a summary of the EPF mechanisms used to implement each type of feature.

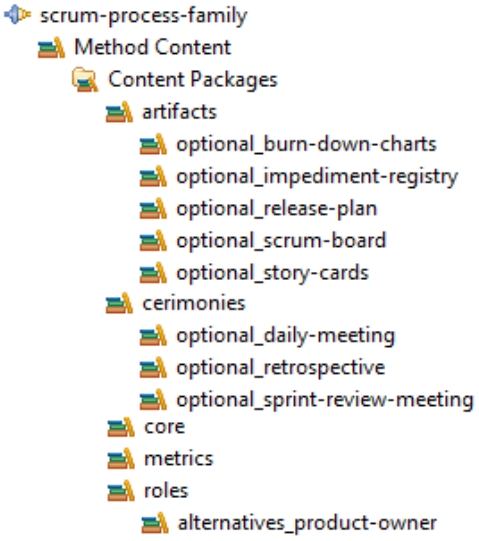

Figure 2. Fragment of the Scrum method plugin process elements

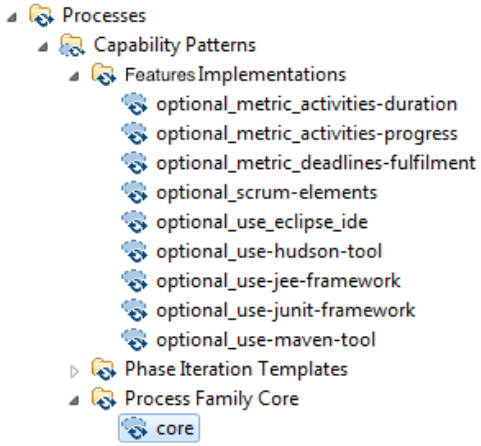

Figure 3. Capability patterns created for the OpenUP process line

Process derivation. To allow the derivation of process instances of the software process lines, the EPF Composer provides the configuration definition functionality. In the configuration definition, a process engineer visually selects the

process structures that will be part of the derived process. However, the EPF Composer does not provide support to specify the process commonalities and variabilities using a representation like a feature model, and their respective mapping – configuration knowledge – to the associated process elements. Because of that, process engineers have to know which selections have to be made to reflect the desired features.

TABLE IV. EPF MECHANISMS USED TO VARIABILITY MANAGEMENT

Type of feature	Adopted EPF mechanism
Alternative	(1) Hierarchical content package structure (2) Process elements representing each alternative (3) Demanded capability patterns encapsulating workflow increments, hierarchically structured
Optional	(1) Content package structure (2) Related process elements (3) Capability pattern encapsulating workflow increments, if needed
OR-feature	(1) Hierarchical content package structure (2) Process elements associated to each option (3) Demanded capability patterns encapsulating workflow increments, hierarchically structured

Figure 4 shows an example of an alternative feature selection. It presents how the selection of an alternative feature is. In this example, the Use Cases feature was selected as the requirements specification technique. Figure 5 presents an example of optional features selection. It shows the selection of five of nine optional features available, which are: (i) the application of a metric for assessing the duration of the main activities; (ii) the inclusion of process elements that address Scrum practices; (iii) the use of Eclipse IDE; and so on. After the definition of the configuration, EPF generates a web site of the requested customized process with its respective workflows and process elements (activities, steps, roles, among others).

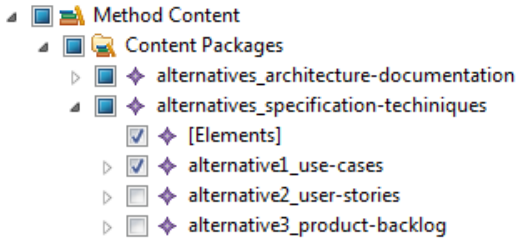

Figure 4. Example of alternative feature selection for OpenUP process line

B. *Process Line Modeling using GenArch-P*

The definition of the software process lines using the GenArch-P required the definition of a new complete process specification. This process specification is composed of: (i) all mandatory process elements that model the core of the process line; and (ii) the optional and alternative process elements associated with specific variable high-level features.

Process line engineering. First, a new GenArch-P project was created for each process line, containing the predefined software process family specifications: OpenUP and Scrum. After that, GenArch-P parsed these specifications and generated a simplified process models for both OpenUP and Scrum. Figure 6 shows the GenArch-P process model generated for the OpenUP based process line, where the elements are hierarchically distributed to compose a wide view of the software process. In order to associate the process elements to specific features, the process engineers must annotate them with features expressions, thus specifying the configuration knowledge. More than one feature can be associated to a given process element. During this activity, the process engineers can also specify constraints and dependencies between features. The modeling of the Scrum process line using the current version of the GenArch-P fails to modularize fine-grained variabilities that happen inside the process elements. The abstraction provided by the simplified process model does not allow the annotation of element content fragments with feature expressions.

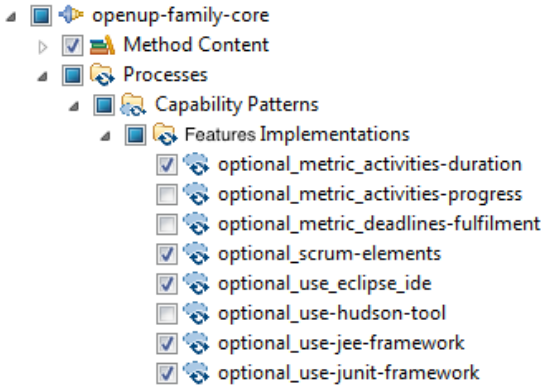

Figure 5. Example of optional feature selection for OpenUp process line

Figure 6 also shows the process elements annotated with their respective associations to the variabilities of the process line. It specifies the configuration knowledge that models the mapping between features and process elements. For example, the feature that specifies the software documentation according with the agile design [14], is associated with the following process elements: (i) concurrent testing; (ii) continuous integration; (iii) test-driven development; (iv) tester; and so on. Figures 7 and 8 show a partial view of the generated feature model for the OpenUP and Scrum process lines, respectively. They model the different variabilities and respective constraints that can be associated to the process elements. Both the feature model and the annotated process model guide the tool during the automated process derivation.

Process derivation. The first step during the automated process derivation using GenArch-P is to create a new feature configuration from the feature model. A feature configuration allows the process engineers to select the desired features for the new software process instance. In GenArch-P, the feature model is specified using the FMP plugin. The FMP plugin makes the features selection a dynamic task, because constraints associated with features are analyzed at the moment that a selection is made. Thus, the selection of specific features may imply the addition or removal of other features. After the feature selection, GenArch-P automatically derives the correspondent software process specification with the process elements associated to the chosen features. The generated software process specification can be then edited and processed by the responsible process engineers.

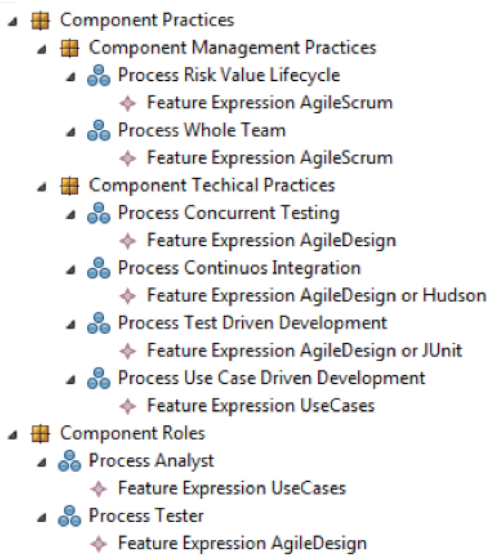

Figure 6. Fragment of the OpenUP family process model with annotations

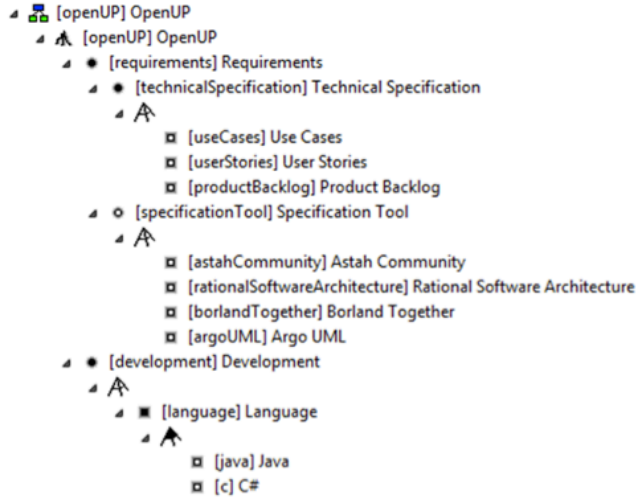

Figure 7. Fragment of the feature model for OpenUP process line

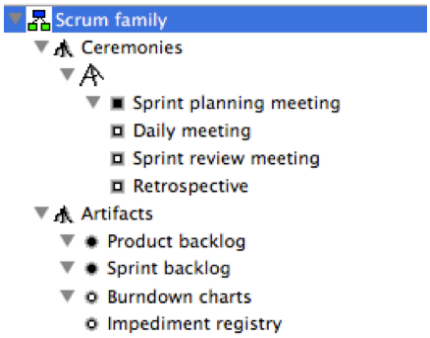

Figure 8. Fragment of the feature model for Scrum process line

C. *Criteria Analysis*

Modularity. Using the EPF Composer it is possible to define software process elements in specific content packages (Figure 1). It is also possible to group flow of activities in a capability pattern, that may be included or not in different configurations - instances of the process line (Figures 3 and 6). These modularization mechanisms do not address the explicit association of process elements to features, but at least, they allow the grouping of process elements related to a given feature. Thus, our study concludes that EPF Composer provides useful support for modularization of process specifications. Table V summarizes some quantitative results for EPF modularization mechanisms.

TABLE V. QUANTITATIVE RESULTS USING EPF COMPOSER

EPF Mechanisms	OpenUP process line	Scrum process line
Core process elements (mandatory)	75	6
Process elements inside content packages (coarse-grained variability)	227	32
Process elements with content variabilities (fine-grained variability)	117	19
Capability patterns	9	0

GenArch-P does not provide modularization mechanisms for the process specification. The process engineers only interact with a simplified process model that aggregates all the process elements. Working with this process model, the process engineers abstract the details and organization of the low-level process specification. They do not need to manage the knowledge about the low level organization of the process line specification, when deriving new processes. The GenArch-P works with a non-modularized process specification. As a consequence, we have concluded that this approach has a weak support for modularization. Table VI presents quantitative results for the GenArch-P after performing similarly tasks as the EPF Composer tasks presented in Table V.

TABLE VI. QUANTITATIVE RESULTS USING GENARCH-P

GenArch-P Mechanisms	OpenUP process line	Scrum process line
Process elements with annotation	**152**	**25**
Process elements without annotation	**75**	**6**

Comparing the quantitative results for modularity mechanisms, it can be seen that although the EPF provides more explicit mechanisms to model the process variabilities, GenArch-P requires to only annotating a very reduced set of process elements (152) compared to the high number of process elements that represent coarse (227) and fine (117) grained variabilities in EPF Composer. In EPF Composer, the higher number of elements is justified because some mandatory process elements are specialized in more than one content package, specifying different variabilities.

Traceability. Using the EPF Composer, the traceability is achieved by the organization of the elements in the correspondent structures: content packages and capability patterns. To facilitate the traceability, the names of these process structures must reflect the name of the correspondent features (Figures 1 and 3). Although the EPF Composer lacks an explicit mechanism to trace from features to process elements, the adequate organization of these elements enables the process line traceability. It shows that EPF Composer has a partial support for traceability.

On the other hand, using GenArch-P the process engineers can visualize all mapping relationships between features and process elements in the annotated process model. GenArch-P

enables us to find all the process elements associated to a specific feature by browsing this model. Due to this characteristic, we can say that the GenArch-P offers good support for traceability.

Error detection. The EPF Composer does not offer an explicit mechanism to detect errors in the derived processes of the process line. For example, it is considered an error the inclusion of two (or more) mutually exclusive features. The only consistency checking performed by EPF Composer is a dependency checking during the configuration process in which it is analyzed if all the method plugins necessary to a specific configuration are properly selected. There is no additional mechanism to check the consistency of the selections made in a process line configuration. For example, in the process configuration definition, it is possible to select two or more options of an alternative feature. The approach does not provide support to guarantee the semantic associated to different kinds of features. Another example of this limitation is that the selection of the JEE Framework optional feature is not bound to the selection of the Java language as one of the alternative features of programming languages to be used during the process execution. It means that EPF cannot specify constraints between features. Because of that, we can conclude that EPF Composer offers a weak support for error detection.

The GenArch-P also does not provide an explicit mechanism for error detection for the process line nor for the derived process instances, but during the annotation of the process elements, it provides support to the definition of constraints associated with the features. These constraints ensure that existing constraints between features will be respected, such as requires or excludes relationships. Thus, we can say that GenArch-P provides a partial support for error and consistency detection.

Granularity. In terms of granularity of the process elements associated with specific features, the EPF Composer allows working at different granularity levels, but with some limitations. The content package and capability pattern mechanisms provide support to coarse-grained granularity through the grouping of process elements. The content variability mechanisms from EPF can be used to append existing attributes to a base process element (with the *contributes* content variability), or redefine existing attributes of a base process element (with *extends* content variability), or the replacement of all its attributes (with *replaces* content variability), thus providing support to fine-grained granularity. It confirms that EPF Composer has a good support for working with coarse and fine-grained process elements. Tables VII and VIII give an overview of the feature granularity for each process lines, and detail the EPF mechanism adopted to model them, which can be used as preliminary guidelines to address those kinds of variabilities.

GenArch-P restricts the granularity of the process elements to the level of elements captured by the parsing of the process specification in order to generate the simplified process model. The first activity of GenArch-P approach is the parsing of the software process definition and the creation of a simplified process model, which will be used by the process engineer to annotate the elements associated with specific features. The current implementation of GenArch-P only supports the granularity of process activities and tasks, but not yet their attributes. It shows that GenArch-P currently provides a partial support for fine-grained granularity.

TABLE VII. GRANULARITY RESULTS FROM MODELING USING EPF

	OpenUP process line	**Scrum process line**
Fine-grained features	0	25
Coarse-grained features	22	0

TABLE VIII. GRANULARITY RESULTS FROM MODELING USING GENARCH-P

	OpenUP process line	**Scrum process line**
Fine-grained features	-	-
Coarse-grained features (element level)	152	25

Comparing the results of Table VII and VIII, one can conclude that the modularization mechanisms of EPF also help the modeling of coarse-grained features. The work with GenArch-P is more tiring because each process element related to a feature has to be individually annotated. The modeling of the Scrum process line indicated that when there is a large amount of fine-grained features, the amount of work is equivalent for both modeling approaches.

Uniformity. EPF Composer fails in terms of uniform support for different software process specifications. It happens because the EPF Composer is designed to be used defining software processes according to the UMA meta-model. This approach has advantages, because the tool can explore all the strengths of the process specification language. The disadvantage is that the tool is tied up to a specific software process definition language. After this observation is concluded that the EPF Composer does not offer uniform support for different forms of software process specification.

On the other hand, GenArch-P works with a simplified process model that abstracts the process specification language. The specified process elements do not have any dependency with the form of the process specification. It is only required the extension of the tool to implement an importing functionality that allows to translate a concrete process specification to the GenArch-P process model. Because of this, it is concluded that the GenArch-P offers the support to the uniformity criterion.

Adoption. The EPF Composer introduces several concepts and mechanisms that need to be known by the process engineers. Examples of such EPF concepts are: method contents, context packages and capability patterns. In addition, they also need to understand the variability mechanisms to address the definition of process variabilities. Last, but not least, the configuration process of EPF processes also requires understanding the functionalities of customization and composition of process elements. Although EPF provides tool support to all these concepts and mechanisms, there is an inherent complexity involved in the adoption of all them to promote the automated variability management of a process line. By forcing a process engineer to know a set of non-trivial concepts, functionalities and mechanisms it is concluded that

the EPF Composer have a weak support for adoption criterion. Our study contributes to improve it with the clear indication of which of these mechanisms can be used to address different kinds of variabilities (Table V).

The GenArch-P tool automates most of the tasks involved in the definition of the software process line. Only few menu selections are needed to start the main functionalities of the tool. First, the tool parses an existing software process specification. The tool then generates a simplified process model. After that, the process engineer annotates the process variable elements, associating them with feature expressions. Finally, the process engineer uses the tool to generate a process instance from process line. In this final step, it is only necessary to create a new feature configuration and the GenArch-P can automatically produces a process instance that reflects those choices. The GenArch-P tool does not require that the user have an extensive knowledge beyond the software process variabilities.

Systematic variability management. Using EPF Composer, the process engineer cannot specify the existing variabilities using a feature model and explicitly associate those variabilities to process elements. The variability management in EPF Composer is only reflected in the modularization and grouping of the process elements representing each of the process line variabilities. On the other hand, GenArch-P allows an explicit feature modeling along all the activities, including the feature representation with its respective constraints, the mapping to existing process elements, and the automatic process derivation based on feature selection. Thus, we can conclude that GenArch-P provides explicit support for a systematic variability management. Table IX presents a summary of the results analysis, with the assigning of a conformance level: "+" (good support), "+/-" (partial support) and "-" (weak or no-support).

TABLE IX. RESULT ANALYSIS SUMMARY

	EPF Composer	GenArch-P
Modularity	+	-
Traceability	+/-	+
Error Detection	-	+/-
Granularity	+	+/-
Uniformity	-	+
Adoption	-	+
Variability Management	-	+

D. Open Issues and Discussions

The study results allowed the observation of existing limitations in the definition of process lines from the investigated approaches. In this section, we discuss some of these limitations and possible solutions to these open issues.

The main limitations of EPF Composer encountered during the study were: (i) it does not provide support to systematic variability management; and (ii) usage complexity due to the diversity and complexity of concepts and mechanisms to allow the modeling of the process line. It is comprehended that the EPF Composer was not designed for implementing software process lines, and this explains the weak support for feature modeling that negatively affects the systematic variability management of the approach. However, the study helped to clarify that even with those limitations, the EPF Composer can be used to modularize software process lines. The introduction of feature modeling and explicit mapping from features to process elements can directly contribute to improve the process line engineering, thus motivating its usage in the software process community. The problem of the usage complexity of the EPF Composer can be overcome with explicit guidelines that show how different process variabilities can be addressed using its abstractions and mechanisms. Our work has also contributed in this sense providing preliminary guidelines to the modularization of software process lines with OpenUP. Although the learning curve is big at first, this effort can be compensated over time.

Similarly, the GenArch-P tool also presented some limitations in the software process line modeling. The most important limitations were: (i) the modularization of variabilities is not addressed, which is an intrinsic characteristic of annotative approaches compensated with the usage of variability analysis and error detection tools; and (ii) the version of GenArch-P used in the study was able to parse only one meta-model for software process specification. The GenArch-P has a simplified usage and automates much of the necessary tasks to the software process line modeling. The problem with the modeling of fine-grained variabilities using GenArch-P can be addressed by improving its parsing and visualization support with the inclusion of new and more fine-grained elements in the process specification, which can be associated with variabilities from the feature model. One of the advantages of the GenArch-P tool is that it abstracts and does not depend on the language or meta-model used to specify the software process. This identified problem can be solved with the implementation of new parsing and visualization modules to other existing ways of process specification.

V. RELATED WORK

Over the last years many research work have been proposed in the context of software process lines. In this section, we present and discuss some of these related works on assessment of modeling approaches for software process lines.

Martínez-Ruiz et al. [6] presents an empirical study to verify whether the variability constructions supported by vSPEM [15] [16] are more suitable for modeling process variability than SPEM 2 [17]. Their study compares the efficiency of the vSPEM variation mechanisms and the understandability of its respective diagrams. It highlights that the biggest problems encountered in modeling variability with SPEM 2 are associated with the fact that we need to apply changes to the method content definition (similar to software libraries in system development) when performing adjustments in process. These changes can affect many different unrelated processes that reuse these content methods. In addition, SPEM 2 does not include a specific notation to model process variability. An alternative to specify variabilities in SPEM 2 is the usage of UML association and inheritance relationships, characterized by stereotypes. The obtained results showed that vSPEM notation was more intuitive and easy to use than the SPEM 2 variability mechanism. However, it is emphasized that additional empirical studies or replications are needed to better evaluate these issues.

Simmonds at al [18] present relevant kinds of software process variabilities that they have found. They discuss how SPEM 2, vSPEM and existing SPL tools and techniques can be used for variability modeling in software processes. The expressiveness of notation for dealing with the required process variability, the understandability of the specification, the adherence to standard formats, and the tool support availability are analyzed. Similar to our work, they claim that SPEM with EPF Composer provide an interesting form-based user interface, but its underlying variability concepts are complex. Regarding vSPEM, they concluded that it is highly promising because it allows the explicit modeling of variabilities from software processes. The authors have not developed any comparison study using both approaches, because the vSPEM is not currently supported by an automated tool.

All these works discuss variability mechanisms and tools focusing at modeling level. However, there are many differences between them and our work. The empirical and comparative studies previously presented did not explore the specification of variabilities from established process frameworks, such as OpenUP and Scrum. In addition, most of the approaches are not assessed from the perspective of variability management and automated process derivation using systematic comparison criteria, as we have addressed in our study.

VI. CONCLUSIONS

This paper presented the results of a comparative study of compositional and annotative process line modeling approaches. Our comparative study was based on a comparison criteria previously adopted in the analysis of SPL implementations [12]. Two modern and recent proposed approaches, named EPF Composer and GenArch-P, were selected to specify non-trivial Open-UP and Scrum process lines. In our study, the GenArch-P annotative approach demonstrated a better results than the EPF Composer compositional approach in terms of most of evaluated criteria, which are: (i) traceability, (ii) error detection, (iii) uniformity, (iv) adoption, and (v) systematic variability management. It is important to emphasize that regarding the *adoption* criterion; it is still fundamental to conduct other studies to understand the usability of process lines modeling approaches. The results cannot be generalized due to the restricted scope of the study. More empirical studies have to be made to achieve additional conclusive results.

The study illustrated that annotative and compositional approaches have their own advantages and limitations when used in the modeling of software process lines. It also concluded that existing compositional mechanisms from EPF could be integrated with annotative approaches specially to improve the modularity of process elements associated to coarse and fine-grained variabilities. In addition, it must be emphasized the great need to enhance the error and consistency detection functionalities of existing annotative approaches. The possible integration of the compositional and annotative approaches can effectively combine the strengths of these two approaches and will be investigated in future work.

ACKNOWLEDGMENTS

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES) – CNPq under grants 573964/2008-4 and CNPQ 560256/2010-8.

REFERENCES

[1] K. Pohl, G.Böckle, and F. van der Linden, Software product line engineering: foundations, principles, and techniques. Berlin, Germany: Springer-Verlang, 2005.

[2] H. Dieter Rombach, "Integrated Software Process and Product Lines," in Unifying the Software Process Spectrum, International Software Process Workshop, Beijing, China, 2005, pp. 83-90.

[3] H. Washizaki, "Building Software Process Line Architectures from Bottom Up," in Product-Focused Software Process Improvement, 7th International Conference, Amsterdam, Netherlands, 2006, pp. 415-421.

[4] T. Ternité, "Process Lines: A Product Line Approach Designed for Process Model Development," in 35th Euromicro Conference on Software Engineering and Advanced Applications, Patras, Greece, 2009.

[5] B. Simidchieva et al, "Representing Process Variation with a Process Family," in International Conference on Software Process, Minneapolis, MN, USA, 2007, pp. 109-120.

[6] T. Martínez-Ruiz et al, "Modelling Software Process Variability: an Empirical Study," IET Software, vol. 5 (2), pp. 172-187, April 2011.

[7] O. Armbrust et al, "Scoping software process lines," Software Process: Improvement and Practice, vol. 14-3, pp. 181-197, May/June 2009.

[8] A. Barreto and et al, "Supporting the Definition of Software Processes at Consulting Organizations via Software Process Lines," in 7th International Conference on the Quality of Information and Communications Technology, Porto, Portugal, 2010, pp. 15-24.

[9] F. Aleixo, M. Freire, W. Santos, and U. Kulesza, "A Model-driven Approach to Managing and Customizing Software Process Variabilities," in 12th International Conference on Enterprise Information Systems, Funchal, Madeira, Portugal, 2010, pp. 92-100.

[10] Eclipse Foundation. (2012) Eclipse Process Framework Project (EPF). [Online]. http://www.eclipse.org/epf/

[11] E.Cirilo and et al, "A Product Derivation Tool Based on Model-Driven Techniques and Annotations," The Journal of Universal Computer Science, vol. 14-8, pp. 1344-1367, 2008.

[12] C. Kästner, Virtual Separation of Concerns: Toward Preprocessors 2.0. Magdeburg, Germany: Otto-von-Guericke-Universität, 2010.

[13] C. Kästner et al, "Integrating Compositional and Annotative Approaches for Product Line Engineering," in GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE), Passau, Germany, 2008.

[14] C. Kästner, S. Apel, M. Kuhlemann, "Granularity in software product lines," in ICSE, 2008, pp. 311-320.

[15] F. Aleixo, M. Freire, W. Santos, U. Kulesza, "An Approach to Manage and Customize Variability in Software Processes," in Brazilian Symposium on Software Engineering, Salvador, Brazil, 2010.

[16] Scott W. Ambler. (2011) Agile Modeling. [Online]. http://www.agilemodeling.com/essays/agileDesign.htm

[17] T. Martınez-Ruiz, F. García, and M. Piattini, "Towards a SPEM v2.0 extension to define process lines variability mechanisms," Software engineering research, management and applications, pp. 115–130, 2008.

[18] T. Martínez-Ruiz, F. García, and M. Piattini, "Enhanced variability mechanisms to manage software process lines," in EUROSPI, Madrid, 2009.

[19] OMG. Software and Systems Process Engineering Metamodel Specification. [Online]. http://www.omg.org/spec/SPEM/2.0

[20] J. Simmonds, M. Bastarrica, L. Silvestre, and A. Quispe, "Analyzing Methodologies and Tools for Specifying Variability in Software Processes ," Computer Science Department, Universidad de Chile, Santiago, Chile, 2011.