Modularizing Software Process Lines using Model-driven Approaches
A Comparative Study

Fellipe A. Aleixo[1,2], Uirá Kulesza[1], Marília Freire[1,2], Daniel Alencar[1] and Edmilson Campos Neto[1,2]
[1]*Department of Computer Science and Applied Mathematics, Federal University of Rio Grande do Norte, Natal-RN, Brazil*
[2]*Federal Institute of Education, Science and Technology of Rio Grande do Norte, Campus Natal-Central, Natal-RN, Brazil*

Keywords: Software Process, Variability Management, Software Process Lines.

Abstract: This work presents a comparative study of the usage of compositional and annotational approaches in the modularization of software process lines. In our comparative study, an Open-UP based software process line extracted from three existing projects are modelled and implemented using the compositional and annotative approaches with the main aim to address a systematic variability management and automatic process derivation. The results show that the GenArch-P – annotative approach – can bring many advantages to the modelling of software process lines considering our comparison criteria.

1 INTRODUCTION

The software process line engineering is inspired in the software product line engineering (Pohl et al., 2005) aiming to promote the large-scale reuse of software processes families. The objective is to address the automatic customization of software processes to specific enterprise contexts and scenarios. Over the last years, several tools and techniques were developed according with these two approaches and several empirical studies have explored and compared their adoption (Kästner & Apel, 2008) (Kästner, 2010). Much of the evolution that happened in the software product line engineering is now being reflected to software processes domain.

The investigation in software process lines have grown in the last years and have reached important results. The consolidation of the theme was achieved by relevant research work, within such areas: (i) motivation for software process line engineering (Rombach, 2005) (Washizaki, 2006) (Ternité, 2009); (ii) representing process variation (Simidchieva et al., 2007) (Martínez-Ruiz et al., 2011) (Simmonds & Bastarrica, 2011); (iii) scoping software process lines (Armbrust et al., 2009); and (iv) modularization of software process lines artifacts (Barreto et al., 2010) (Aleixo et al., 2010). With the consolidation of this research field, other challenges emerge and need to be addressed by researchers. Examples of existing challenges related with the software process line modularization are: Which approaches for process modularization are available? Which techniques and tools should be used in different scenarios? Which are the benefits and limitations of each of these approaches?

This work presents a comparative qualitative study of the usage of compositional and annotational approaches in the modularization of software process lines. The compositional approach is represented by the EPF Composer (EPF, 2012) an industrial process engineering tool that supports the modularization and composition of process elements. The annotational approach is represented by the GenArch-P (acronym for GenArch-Process) (Aleixo et al., 2010), an adaptation of an existing model-based product derivation tool (Cirilo et al., 2008) that provides support to create generative models that represents the process variabilities and associate them with existing process elements.

Our work adapted the comparison criteria from Kästner (2008) (2010) to analyze these approaches from different perspectives, such as: *modularity*, *traceability*, *error detection*, *granularity, uniformity* and *adoption*. Beyond these criteria is included a new criterion that is the support to *systematic variability management*.

The remainder of this paper is organized as follows. Section 2 provides an overview of the two

120 A. Aleixo F., Kulesza U., Freire M., Alencar D. and Campos Neto E..
Modularizing Software Process Lines using Model-driven Approaches - A Comparative Study.
DOI: 10.5220/0004005801200125
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 120-125
ISBN: 978-989-8565-11-2
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

investigated approaches. Section 3 describes the proposed case study, the target software process line and the used comparison criteria. Section 4 describes the case study realization and the obtained results. Finally, the conclusions are presented in Section 5.

2 MODEL-DRIVEN APPROACHES FOR SOFTWARE PROCESS LINES

In recent 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 modularization of process variabilities. One of these techniques is EPF that allows modularizing process elements using compositional refinement techniques. The other one is called GenArch-P that promotes the variability management of process artifacts using annotation-based techniques.

2.1 Compositional: EPF Composer

The EPF Composer is a conceptual framework for authoring, tailoring and deployment of software processes (EPF, 2012). In EPF, a sofware process is organized in terms of method content and processes Method contents describe the process elements, such as: activities, roles, practices, guidances, among others. Processes describe the flow of activities, which reference the elements defined in method content. Process variabilities in EPF can be handled with four variability mechanisms that can be applied to method contents and processes. Those mechanisms are (EPF, 2012): (i) *contributes*, (ii) *replace*, (iii) *extends* and (iv) *extends and replace*.

In *contributes* variability mechanism, a specialized element appends the content of a base element. In *replaces* variability mechanism, a specialized element takes place of a base element. In *extends* variability mechanism, a specialized element reuses some attributes, and overwrite others of the base element. At last, the *extends and replace* variability mechanism combines the effects of the *extends* and the *replaces* variability mechanisms into one strategy. While the *replace* variability replaces all contents from the base element, this variability mechanism allow the inheriante of the non-defined attributes of the specialized element.

2.2 Annotation: GenArch-P

GenArch-P (Aleixo et al., 2010) is a tool for software process derivation adapted from its original version to software product lines, called GenArch (Cirilo et al., 2008). The tool is based on the annotational approach, where the variable elements of a software process are annotated to reflect which type of variation is implement by the element. The annotational approach is widely used in various software product lines tools, such as pure::variants and CIDE.

The GenArch-P works with two models: a simplified process model and the feature model. The elements of the process model are annotated to reflect their relationship to specific features. 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 variation type of this element and the feature parent.

The process model is a simplified representation of the software process specification, that presents the process elements as a hierarchical tree. The feature model is automatically generated by the tool. After the annotations, the process model acts like a configuration model, allowing the visualization of the associations between features and process elements. The feature model is a common artifact at the software product lines development, introduced by Feature-Oriented Domain Analysis (Kang et al., 1990), that represents all the possible instances of a product line. The model presents all the possible options to be selected, in terms of features.

3 STUDY SETTINGS

In our study, we have modelled the same OpenUP-based process line composed of different kinds of process variabilities using the evaluated modelling approaches. After the specification of the software process line using EPF and GenArch-P, we have conducted a comparative analysis of the final results based on criteria adapted from a previous study (Kästner et al., 2008) (Kästner, 2010).

3.1 Target Software Process Line

The definition of the software process line used in our study involved the analysis of OpenUP-based processes from three existing research and development projects. These projects were developed in cooperation between our institution and other ones. The first project was the development of a software system for auditing the 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 complete specification of the software process line to be modularized was accomplished using the extractive technique. It comprised the analysis of the commonalities and variabilities of the selected software processes. The common process elements compose the core of the process line, while the variable process elements were modelled separately and associated to high-level features. It was identified 76 process elements composing the core of the process line. Regarding the variabilities, it was found 9 optional features, 8 alternative features, and 5 OR-features. Table 1 shows the names of the identified features and the number of associated process elements.

Table 1: Identified features in the software process family.

Feature	Alternatives (if there is)	Process elements
Others processes influences		
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	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 – SVN mined)		5
Metric for assess the activities progress		3
Metric for asses the deadlines fulfilment		3
Metric for asses 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 requires the selection of Java features as the programming language. Further information about the specification of the software process line using the different approaches can be found here (Aleixo et al., 2012).

3.2 Comparison Criteria

In order to promote the assessment of software product line implementation techniques, Kästner et al. (2010) have defined comparison criteria with such purpose. In our work, we have adapted most of these criteria – *modularity*, *traceability*, *error detection*, *granularity*, *uniformity*, and *adoption* – to the context of software process lines, which are presented in Table 2. In addition, we have also analysed the approaches support for systematic variability management.

Table 2: Comparison criteria.

Criterion	Definition
Modularity	It analyses the support to modularization of the process elements associated with specific features (features implementations), isolating the implementation of a specific feature
Traceability	It shows how easy is the visualization of the mapping between features and their related process elements
Error detection	It analyses how existing approaches provide support to consistency checking of the software process line and their (resultant) derived processes
Granularity	It assesses the approach support to associate features with process elements of different granularity, considering also the attributes of the process elements
Uniformity	It analyses the ability to uniform support for different forms of software processes specification, evaluating how an approach is tied to a specific process specification form
Adoption	It determines the difficulty of adopting existing approaches in terms of the amount of necessary knowledge (concepts, mechanisms and tools) for the approach application
Systematic variability management	It assesses the approach mechanisms for systematic and effective management of variability: (i) the variability specification, their constraints, and the mapping with process elements; and (ii) the approach support to automated process derivation from existing process core assets

4 STUDY RESULTS

In this section, we report our study results by describing the process line modelling using EPF Composer (Section 4.1) and GenArch-P (Section 4.2). In addition, we also present and discuss the obtained results for the comparison criteria of the two approaches (Section 4.3).

4.1 Process Line Modelling using EPF Composer

Process Line Engineering. The modelling of the process line was accomplished taken as basis the original method plugin of OpenUP. EPF defines a customizable software process in terms of *method contents* and *processes*, which are fundamental concepts from EPF. The *method contents* allow specifying the process elements and all their attributes, which can later be (re)used and composed to define the workflows of a new process.

The process line engineering involved the following steps: (i) creation of a new method plugin; (ii) creation of the mandatory process elements, in a "core" content package; (iii) creation of specific content packages to each feature, with the correspondent process elements; (iv) creation, if necessary, of the correspondent capability patterns to each feature, to encapsulate the additions to the workflow defined by the "core" elements. Each of these capability patterns uses the content variability "contributes" mechanism to refine the capability pattern that represents the core including the specific flow associated with the process variability. Table 3 presents a summary of the EPF mechanisms used to implement each type of feature.

Table 3: EPF mechanisms used to each type of feature.

Type of feature	Used EPF mechanism
Alternative	(i) hierarchical content package structure + (ii) process elements of each alternative + (iii) capability patterns to encapsulate the additions to the "core" workflow
Optional	(i) content package structure + (ii) associated process elements + (iii) capability patterns to encapsulate the additions to the "core" workflow
OR-feature	(i) hierarchical content package structure + (ii) process elements of each related option + (iii) capability patterns to encapsulate the additions to the "core" workflow

Process Derivation. EPF Composer provides the functionality to the definition of a process configuration. In the configuration definition, the process engineer chooses which modular structures will be part of a published process – a navigable web site. The process of configuration definition and process instance publication can be repeated, allowing the publication of all process line possibilities. After the configuration definition, a customized process is published with its respective workflows and process elements (activities, tasks, roles, among others).

4.2 Process Line Modeling using GenArch-P

Process Line Engineering. The process line engineering involved the following steps: (i) definition of a complete software process specification (ii) creation of a new GenArch-P project; (iii) use the tool to parse the process specification, generating a simplified process model; (iv) annotation of the involved process elements with features expressions defining the configuration knowledge; (v) specification of constrains and dependency relationship between features.

Figure 1 shows an example of the GenArch-P process model where the elements are hierarchically distributed to compose a wide view of the software process structure. Figure 1 also shows the annotations in process elements defining associations to the variabilities of the process line. It represents the configuration knowledge of mapping from features to process elements. For example, the feature that represents the agile design alternative feature is associated to the following process elements: (i) concurrent testing, (ii) continuous integration, (iii) test-driven development, and others not presented in the figure. After the annotation of process elements, the GenArch-P generates the correspondent feature model. Both the feature model and the annotated process model guide the tool in the automated process instance derivation.

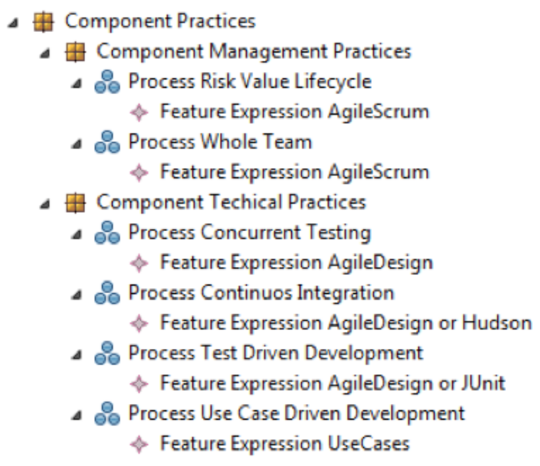
Component Practices
Component Management Practices
Process Risk Value Lifecycle
Feature Expression AgileScrum
Process Whole Team
Feature Expression AgileScrum
Component Techical Practices
Process Concurrent Testing
Feature Expression AgileDesign
Process Continuos Integration
Feature Expression AgileDesign or Hudson
Process Test Driven Development
Feature Expression AgileDesign or JUnit
Process Use Case Driven Development
Feature Expression UseCases

Figure 1: Fragment of the process model with the features annotations.

Process Derivation in GenArch-P starts with the creation of a new feature configuration that allows the processes engineers to select the desired features for a new process instance. During the features selection, constraints associated with features are analysed. Thus, the selection of specific features may imply the removal of others. After the features selection, a new process specification is derived, with the process elements associated to the chosen features.

4.3 Criteria Analysis

Modularity. Using EPF Composer is possible to group software process elements in specific content packages, and workflows in capability patterns. These modularization mechanisms allow the grouping of process elements related to a given feature. Thus, our study concludes that EPF Composer provides useful support for modularity of process specifications. GenArch-P does not provide modularization mechanisms for the process specifications. The process engineers only interact with a simplified model of the process, abstracting the details and organization of the process specification. We concluded that GenArch-P has partial support for modularization, because the software process specification could already be modularized with low-level mechanisms.

Traceability. Using the EPF Composer the traceability is achieved by the organization of the elements in the correspondent structures: content packages and capability patterns. Although the EPF Composer lacks an explicit mechanism to map features to process elements, we can conclude that EPF Composer has a partial support for traceability. Using GenArch-P, the process engineer can visualize all mapping relationships between features and process elements in the annotated process model. Due to this characteristic, it is concluded that GenArch-P offers good support for traceability.

Error Detection. The EPF Composer does not offer a mechanism to detect semantic errors in the specification of the process line. In addition, EPF cannot represent constraints between features. The only initiative accordingly is a dependency checking during the configuration definition. Because of that, we can conclude that the EPF Composer offers a weak support for error detection. GenArch-P does not have an explicit mechanism for error detection, but during the annotation of the process elements, it can specify constraints associated with the features. These constraints allow guaranteeing that existing constraints between features will be respected, such as requires or excludes relationships. Thus, we can say that GenArch-P provides currently only partial support for error detection.

Granularity. The content package and capability pattern mechanisms from EPF Composer provide support to coarse-grained granularity through the grouping of process elements and workflows. Moreover, the variability mechanisms can be used to redefine existing attributes of process elements, thus providing support to fine-grained granularity. We conclude that EPF Composer has a good support for coarse and fine-grained elements. GenArch-P restricts the granularity of the process elements at the level of elements captured by the parsing of the process specification in order to generate the simplified process model. The current version of GenArch-P only supports the granularity of process activities and tasks, but not yet their attributes. We concluded that GenArch-P currently provides a partial support for fine-grained granularity.

Uniformity. EPF Composer does not provide uniform support for different software process specifications. It happens because the EPF Composer only specifies software processes according with the UMA meta-model. Because of that, it was concluded that the EPF Composer does not offer uniform support for different forms of software process specification. Using GenArch-P, the process specification language is abstracted by a simplified process model. This model has not any dependency with the low-level process specification. Because of this, it is concluded that the GenArch-P provides support to the uniformity criterion.

Adoption. The EPF Composer provides an ample set of concepts and mechanisms that need to be known by the process engineers. In addition, they also need to understand the variability mechanisms to address the modularization and configuration of process variabilities. By forcing a process engineer to know a set of mechanisms, concepts and functionalities, it is concluded that EPF Composer has weak support for the adoption criterion. The GenArch-P tool automates most of the tasks involved in the modelling of the software process line. It does not require that the process engineers have an extensive knowledge beyond the software process variabilities, allowing an easy adoption.

Systematic Variability Management. Using EPF Composer, the process engineer cannot specify the existing variabilities using a feature model and explicitly associate them to process elements. On the other hand, GenArch-P allows an explicit feature modelling, 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 offers a systematic variability management.

5 CONCLUSIONS

This paper presented the results of a comparative study of compositional and annotational modeling approaches of software process lines. Two modern approaches were selected: (i) EPF Composer – representing the compositional approach and (ii) GenArch-P – representing the annotational approach. These two investigated approches were used to specify a non-trivial Open-UP processes line.

Our study adopted a comparison criteria previously adopted in the analysis of the implementation techniques of software product lines (Kästner, 2010). Based on the results of the study, it can be concluded that the annotational approach obtained better results in the software processes lines definition. In five of the seven defined criteria, the GenArch-P presented better results, which are: (i) traceability, (ii) error detection, (iii) uniformity, (iv) adoption, and (v) systematic variability management. The EPF Composer had better results in the modularity criterion, which reinforces one of the known strengths of compositional approaches. In the granularity criterion, the EPF Composer approach had also better results, due to the variety of variability mechanisms provided.

The study illustrated that annotative and compositional approaches have their own strengths and limitations defining software process lines, and both are valid alternatives. The possible integration of the compositional and annotative approaches can combine the strengths of these two approaches and will be investigated in future work.

ACKNOWLEDGEMENTS

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

Aleixo, F. A., Freire, M. A. & Kulesza, U., 2012. Software Process Lines. [Online] Available at: https://sites.google.com/site/softwareprocesslines/ [Accessed 27 January 2012].

Aleixo, F. A., Freire, M. A., Santos, W. C. d. & Kulesza, U., 2010. A Model-driven Approach to Managing and Customizing Software Process Variabilities. *In 12th ICEIS*. Funchal, Madeira, Portugal, 2010. SciTePress.

Aleixo, F. A., Freire, M. A., Santos, W. C. d. & Kulesza, U., 2010. Automating the Variability Management, Customization and Deployment of Software Processes: A Model-Driven Approach. *Lecture Notes in Business Information Processing*, pp.372-87.

Armbrust, O. et al., 2009. Scoping software process lines. Software Process: Improvement and Practice, 14-3, pp.181-97.

Barreto, A., Duarte, E., Rocha, A. R. & Murta, L., 2010. Supporting the Definition of Software Processes at Consulting Organizations via Software Process Lines. *In 7th QUATIC. Porto, Portugal, 2010. IEEE Computer Society*.

Cirilo, E., Kulesza, U. & Lucena, C. J. P. d., 2008. A Product Derivation Tool Based on Model-Driven Techniques and Annotations. *The Journal of Universal Computer Science*, 14-8, pp.1344-67.

EPF, 2012. Eclipse Process Framework Project (EPF). [Online] Available at: http://www.eclipse.org/epf/ [Accessed 27 January 2012].

Kang, K. C. et al., 1990. Feature-oriented domain analysis (FODA) feasibility study. *SEI*.

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

Kästner, C. & Apel, S., 2008. 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. University of Passau.

Kästner, C., Apel, S. & Kuhlemann, M., 2008. Granularity in software product lines. *In ICSE*., 2008.

Martínez-Ruiz, T., García, F., Piattini, M. & Münch, J., 2011. Modelling Software Process Variability: an Empirical Study. *IET Software*, 5 (2), pp.172-87.

Pohl, K., Böckle, G. & Linden, F. v. d., 2005. Software product line engineering: foundations, principles, and techniques. Berlin, Germany: Springer-Verlang.

Rombach, H. D., 2005. Integrated Software Process and Product Lines. *In ISPW*. Beijing, China, 2005. Springer.

Simidchieva, B. I., Clarke, L. A. & Osterweil, L. J., 2007. Representing Process Variation with a Process Family. *In ICSP*. Minneapolis, MN, USA, 2007. Springer.

Simmonds, J. & Bastarrica, M. C., 2011. Modeling Variability in Software Process Lines. Santiago, Chile: Universidad de Chile.

Ternité, T., 2009. Process Lines: A Product Line Approach Designed for Process Model Development. *In 35th Euromicro Conference on Software Engineering and Advanced Applications*. Patras, Greece, 2009. IEEE Computer Society.

Washizaki, H., 2006. Building Software Process Line Architectures from Bottom Up. *In PROFES*. Amsterdam, The Netherlands, 2006. Springer.