OdysseyProcessReuse
A Component-based Software Process Line Approach

Eldânae Nogueira Teixeira[1], Aline Vasconcelos[2] and Cláudia Werner[1]
[1]*Systems Engineering and Computer Science Program, COPPE/UFRJ, ZIP Code: 21945-970, P.O. Box 68511, Rio de Janeiro, RJ, Brazil*
[2]*Federal Fluminense Institute of Education, Science and Technology, Campos dos Goytacazes, RJ, Brazil*

Keywords: Software Process Reuse, Software Process Line, Software Process Component.

Abstract: It is expected that managing process variations and organizing process domain knowledge in a reusable way can provide support to handle complexity in software process definition. In this context, the purpose of this paper is to describe a systematic software process reuse methodology, by combining process reuse techniques, such as Software Process Line and Component Based Process Definition, aiming to increase reuse possibilities. SPrL approach manages the variability aspect inherent to software process domain and CBPD focuses on modularizing the domain process information into process components. The proposed SPrL modelling metamodel and notation address reusable process elements, explicitly representing the variability concept in both process domain structure and behaviour. Based on the results of the evaluation studies, it was possible to get evidences of the approach feasibility, with a higher expressiveness when using the process variability notation proposed, which allow that more semantic concepts inherent to SPrL scenarios can be graphically described. Also, the set of heuristics to support mappings among artefacts in distinct abstraction levels was considered useful to keep the traceability of variability properties, relationships and restrictions. Further research is being conducted to explore ways to support project managers during the decision-making in new software process definitions.

1 INTRODUCTION

One way for defining a process can be to apply process tailoring approaches. Software process tailoring is "the act of adjusting the definition and/or particularizing the terms of a general description to derive a description applicable to an alternate (less general) environment" (Ginsberg and Quinn, 1995).

In this scenario, organisations normally adopt an ad hoc tailoring approach, where the intuition and expertise of an experienced project manager or process designer are always involved (Zakaria et al., 2015). It demands experience and involves knowledge from several aspects of software engineering, which usually requires a highly skilled professional who is able to reconcile all these factors (Barreto et al., 2011).

Conventional tailoring approaches can be divided into two major types: component-based approaches and generator approaches (Washizaki, 2006). The former tries to build a project-specific process based on existing process parts, but it lacks a way to address the overall compatibility and consistency of the derived processes and the latter tries to build a project-specific process by instantiating a process architecture, but it lacks a way to reuse process fragments (Washizaki, 2006).

In this context, Software Process Line (SPrL) (Rombach, 2013) has emerged as an approach for software process reuse, based on the concepts of Software Product Line (SPL) (Northrop, 2002). The concepts of Component Based Development (CBD) (Sametinger, 1997) have been applied by approaches that conduct the definition of software processes using components as a Component Based Process Definition (CBPD) (Gary and Lindquist, 1999).

Although several initiatives regarding software process tailoring in software processes exist in the literature, there is no standard approach or consensus regarding how to perform process tailoring in a controlled and consistent manner nor there is a complete notation that supports it (Martínez-Ruíz et al., 2012). There is no current consensus on a standard notation to support process tailoring (Pillat

Nogueira Teixeira, E., Vasconcelos, A. and Werner, C.
OdysseyProcessReuse.
DOI: 10.5220/0006672902310238
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 231-238
ISBN: 978-989-758-298-1

et al., 2015) and we lack the ability to capture required flexibility of software processes due to a missing ability to express flexibility using current process modelling languages (Kuhrmann, 2014).

Also, although different approaches contribute to the componentization strategy of reusable process elements, as stated by Aoussat et al. (2010), even if most existing approaches advance to the same definition for a software process component, no consensus or metamodel describing the software process component characteristics is achieved. Each component definition is based on the intended use of a particular approach and there is a lack of techniques guiding the components development.

So, if knowledge belonging to experienced process engineers could be made explicit, formalized, and available to other professionals, it would probably be possible to reuse this knowledge in an effective way (Barreto et al., 2011). It is expected to minimize rework on process definitions and the need of most experienced process engineers.

Considering this scenario, a systematic software process reuse approach is presented in order to address some of the challenges described above. It combines SPrL and CBPD techniques, aiming to address two aspects involved in the organization of process domain reusable information: (1) process domain variability, and (2) process domain modularity. The approach includes: (i) methodology for SPrL development; (2) process variability representation; (3) domain information modularity treatment; (4) the semi-automation of some steps of Software Process Domain development, aiming to reduce its definition effort; and (5) a supporting environment. The process reuse support deals with how to organize the reusable information in a comprehensive way, where the knowledge required is explicitly represented and it is expected to balance the domain amount of information complexity by using components to organize the domain in a modular way, improving its understandability, maintainability, and ultimately its reusability.

Following this Introduction, the rest of the paper is organised in four sections, namely: Section 2 presents the concepts involved in background knowledge and related work. The method is described in Section 3, contextualizing an overview of the variability modelling proposed. The approach had been evaluated and gradually refined through a set of evaluation studies, presented in Section 4, where the SPrL representation and mapping heuristics evaluation studies briefly describe the approach feasibility. Finally, Section 5 presents the conclusions, ongoing and future work.

2 BACKGROUND AND RELATED WORK

This section introduces the concepts of SPrL and software process components and presents the related work.

2.1 Software Process Reuse

Given the diversity of aspects involved in software development, there is no single framework and guideline that can be used to define the software process in all project environments (Zakaria et al., 2015). In order to cope with this diversity, adaptations according to the context of projects and teams, besides the reuse of past experiences in the definition of new software processes, are needed (Magdaleno et al., 2015). Several approaches were proposed to improve reusability of software processes, and to allow defining methods for composition of customized processes (Schramm et al., 2015). Reusing software processes is important for many reasons, such as: reducing costs and time; increasing quality; promoting expert knowledge reuse; and making process definition accessible to less experienced people (Magdaleno et al., 2015).

In this context, software reuse concepts have been applied for supporting processes reuse (Kellner et al., 1996), emerging approaches as SPrLs and the definition of software processes by using smaller and reusable units, called as process components.

Variability management is a key requirement in the development of SPrLs, providing support to specification, implementation, variability resolution and customized processes generation (Dias and Oliveira Junior, 2016). It is necessary that process modelling languages, and therefore their corresponding metamodels, include variability constructors (Martínez-Ruíz et al., 2011).

Much research has addressed the SPrL topic, but still it can be considered an immature area, since there is not a well-defined taxonomy and the quality assessment of the proposed approaches needs improvement in terms of empirical validation (De Carvalho et al., 2014). Also, a problem with the realization of software process lines is the lack of a common understanding of what is considered to be a process line in practice (Kuhrmann et al., 2016).

Another relevant approach is process definition by composition based on smaller units called process components. It is pointed out as a relevant aspect by reference models and standards, such as MPS.BR (Softex, 2016) and CMMI (Chrissis, 2006). Although different approaches contribute to the componentization strategy of process elements, no consensus or metamodel describing software process component characteristics is yet available (Aoussat et al., 2010). Also, there is not a general agreement on which information has to exist in a process component or which level of detail it must have (Barreto et al., 2011). These approaches do not provide support to the components organization in architectures and decisions are usually based on their intended use in each approach. Also, there are not components grouping techniques to deal with coupling and granularity.

2.2 Software Process Variability Modelling

Software Process Modelling Languages (SPMLs) have been created from different sources and aiming to address different problems, including software process reuse (García-Borgoñon et al., 2014). Due to the large number of SPMLs users, it is difficult to establish the best language to be used and software-intensive organizations have not yet adopted any of the proposed ones in a practical sense, nor there is a basis or standard (García-Borgoñon et al., 2014).

A set of SPrL works has been analysed in order to identify process variability modelling proposals: (1) Software and Systems Process Engineering Metamodel (SPEM 2.0) (OMG, 2008) defines the variability representation by four types of relations: contributes, replaces, extends and extends-replaces; (2) vSPEM approach (Martínez-Ruíz et al., 2008) was proposed as an extension of SPEM 2.0; (3) Stereotype based Variability Management proposal for SPEM (SMartySPEM) (Dias and Oliveira Jr, 2016) aims to support identification and representation of variability in SPEM-based software process elements; and (4) Casper approach (Alegría and Bastarrica, 2012) presents the SPrL representation by three models: contextual model, feature model and process model with variability represented by eSPEM, an extension of SPEM 2.0.

None of the representations proposed can fully meet the needs for process variability modelling, still missing issues such as: process elements variable in a process family (i.e., activity, task, role, work product, tool, and relations) and variation in control flows are not completely addressed; the variability and optionality of process elements cannot be defined together; only few studies propose notations to represent variability in different perspectives (Martínez-Ruíz et al., 2012); and still there is the need for empirical studies and evaluations of proposals (Oliveira Jr et al., 2013). Also, there is a lack of domain complexity treatment by different abstraction levels and the provision of a mapping support among them.

To address these topics, a software process reuse approach is proposed as described in the next section.

3 A SYSTEMATIC SOFTWARE PROCESS REUSE APPROACH

A systematic process reuse approach is proposed and involves the definition of a SPrL Engineering composed by five main elements: i) a method, which is a Process Domain Engineering process combined with CBPD; ii) a representation to variability modelling, addressing two abstraction levels; iii) a mapping mechanism as a guide to trace properties throughout different domain artefacts; iv) a components' grouping technique that aims to address coupling and granularity properties; and v) a supporting environment (Odyssey, 2017).

The Process Domain Engineering process is composed by two main phases (Figure 1): Software Process Domain Engineering (SPDE) - phase where an infrastructure to systematize reuse is conceived and it is the main focus of the proposed approach - and Software Process Project Engineering (SPPE) - phase where project specific processes are derived using the reusable process domain artefacts produced by SPDE.

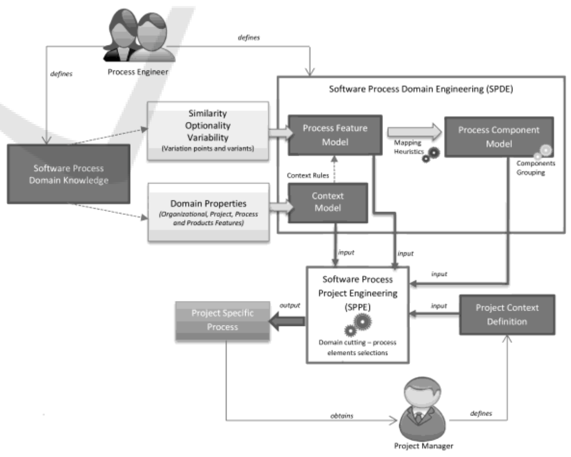

defines
defines
Process Engineer
Software Process Domain Engineering (SPDE)
Similarity
Optionality
Variability
(Variation points and variants)
Process Feature Model
Mapping Heuristics
Process Component Model
Components Grouping
Software Process Domain Knowledge
Context Rules
Domain Properties
(Organizational, Project, Process and Products Features)
Context Model
input
input
input
Software Process Project Engineering (SPPE)
Domain cutting – process elements selections
output
Project Specific Process
input
Project Context Definition
obtains
defines
Project Manager

Figure 1: Process Domain Engineering overview.

3.1 Software Process Domain Engineering (SPDE)

During the SPDE phase, the following main activities are covered: (1) Domain Identification, (2) Domain Knowledge Acquisition, (3) Domain

Similarity and Variability Analysis, and (4) Domain Modelling.

The first activity includes the scope definition that involves a domain feasibility study. Reuse opportunities are investigated considering the current and future organizational strategic goals. The main objective is to identify an appropriate domain with the delimited scope, according to time and resources available versus the expected results.

The second activity aims to capture information and knowledge within the software process domain identified. It is divided into the following tasks: (1) knowledge source(s) identification; and (2) knowledge capture and storage. These tasks may vary depending on the approach applied: bottom-up or top-down and possibly even a combination of both.

The third activity determines points where the domain processes are similar (mandatory elements) and points where they diverge (optional and alternative elements), which represent adaptation points during specific process derivation.

The forth activity results in the final models for a SPrL: (1) Process Domain Feature Model with compositional rules; (2) Projects Context Model with context rules that make the link between these two first models, and (3) Process Domain Component Model, generated by applying mapping heuristic, as described in Section 3.1.2 (Figure 2).

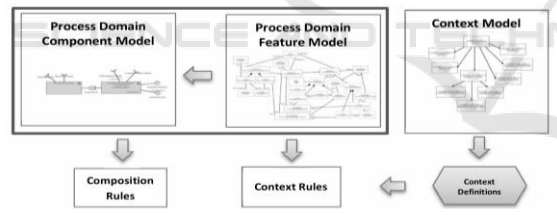


Figure 2: Process Domain Analysis Models.

The process domain knowledge is represented using the metamodel and notation for variability management developed, called *OdysseyProcess-FEX* (Teixeira et al., 2015). This metamodel defines an abstract syntax for two levels of abstraction: (1) feature model, and (2) component model. It was specified by analysing different process models, such as OpenUP and RUP, SPEM 2.0 (OMG, 2008), a process ontology (Falbo and Bertollo, 2005), the process variability modelling literature - briefly presented in Section 2.2, and feature modelling in SPL approaches, specifically *Odyssey-FEX* (Fernandes and Werner, 2008).

A Domain Contextual Model may be defined to represent project properties that can affect processes derivations. It is modelled using *Ubi-FEX* notation, as described in Fernandes and Werner (2008), considering the process feature model defined.

A checklist-based inspection technique was also proposed to the verification of syntactic and semantic feature and component models, called PVMCheck (Teixeira et al., 2015).

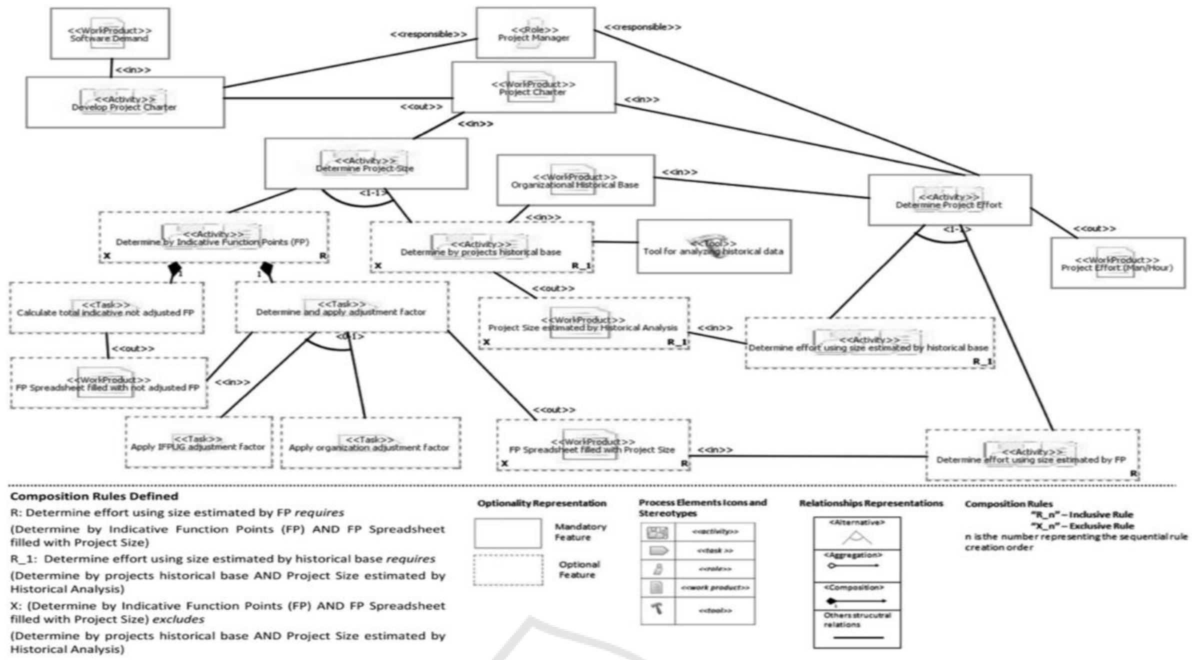
3.1.1 Process Feature Modelling

The Process Feature Modelling represents a process domain conceptual model, i.e., a high-level analysis abstraction of the organizational process and its variations. This includes a domain variability structural representation and a behavioural one, which establishes control flows, indicating the execution order variations among work units. This feature modelling includes the following activities, which can be conducted in parallel:

(1) Represent Process Elements (activities and tasks, roles, work products and tools);
(2) Determine Optionality - Classify each element as mandatory or optional;
(3) Determine Variability - Classify each element as invariant, variation point or variant;
(4) Determine for each variation point its alternatives of configuration (related variants);
(5) Determine structural relations and their optionality property, using Association, Aggregation, Composition among work units, roles, work products and tools relationships;
(6) Define dependency and mutual exclusivity relationships by inclusive and exclusive composition rules, respectively; and
(7) Determine behavioural relations and their optionality property (define control flows and their variations).

Each element and relation is represented by a specific graphical element with icons and stereotypes, as described in Figure 3. This figure also presents an excerpt of a Project Planning SPrL.

This SPrL describes three initial mandatory planning activities: Develop Project Charter, Determine Project Size and effort. The effort can be defined based on of two different techniques for project size determination (Function Points (FP) or by applying a historical base), characterizing a variation point with two variants classified as optional and mutual exclusive and dependent on other elements in the domain. The dependences among elements are defined by composition rules identified by R and R_1 stereotypes in Figure 3.


<<WorkProduct>> Software Demand
<<responsible>>
<<Role>> Project Manager
<<responsible>>
<<in>>
<<Activity>> Develop Project Charter
<<WorkProduct>> Project Charter
<<out>>
<<in>>
<<in>>
<<Activity>> Determine Project Size
<<WorkProduct>> Organizational Historical Base
<<in>>
<<Activity>> Determine Project Effort
<i-1>
<<in>>
<<Activity>> Determine by Indicative Function Points (FP)
<<Activity>> Determine by projects historical base
<<Tool>> Tool for analyzing historical data
<i-1>
<<out>>
<<WorkProduct>> Project Effort (Man/Hour)
<<Task>> Calculate total indicative not adjusted FP
<<Task>> Determine and apply adjustment factor
<<out>>
<<WorkProduct>> Project Size estimated by Historical Analysis
<<in>>
<<Activity>> Determine effort using size estimated by historical base
<<out>>
<<WorkProduct>> FP Spreadsheet filled with not adjusted FP
<<in>>
<i-1>
<<out>>
<<Task>> Apply IFPUG adjustment factor
<<Task>> Apply organization adjustment factor
<<WorkProduct>> FP Spreadsheet filled with Project Size
<<in>>
<<Activity>> Determine effort using size estimated by FP
X
R
R_1
Optionality Representation
Mandatory Feature
Optional Feature
Process Elements Icons and Stereotypes
<<activity>>
<<task>>
<<role>>
<<work product>>
<<tool>>
Relationships Representations
<Alternative>
<Aggregation>
<Composition>
Others strucutral relations
Composition Rules
"R_n" – Inclusive Rule
"X_n" – Exclusive Rule
n is the number representing the sequential rule creation order

Composition Rules Defined

R: Determine effort using size estimated by FP *requires*

(Determine by Indicative Function Points (FP) AND FP Spreadsheet filled with Project Size)

R_1: Determine effort using size estimated by historical base *requires*

(Determine by projects historical base AND Project Size estimated by Historical Analysis)

X: (Determine by Indicative Function Points (FP) AND FP Spreadsheet filled with Project Size) *excludes*

(Determine by projects historical base AND Project Size estimated by Historical Analysis)

Figure 3: Project Planning SPrL Feature Model (Adapted from Teixeira (2016)).

3.1.2 Process Components Modelling

A process component, in our work, can be understood as a process fragment abstraction based on the "black box" principle, representing a modular part of a process that encapsulates its contents and communicates with its environment by interfaces. Each component is composed by work units (activity and task) that represent its possibilities of realization. Four aspects can classify a process component orthogonally: (i) variability (variation point, variant or invariant), (ii) optionality (mandatory and optional), (iii) internal structure (simple or composed by other components), and (iv) internal variation (concrete or abstract). A process component can be considered concrete when there is no kind of internal variation to be resolved, i.e., all its process elements are defined to be directly instantiated and enacted during a process execution with no remaining decisions, otherwise, it is classified as abstract.

The relations among components are established through interfaces. An interface can be of two types: data flow or control flow. The Data Interface represents relations between work units and work products that are exchanged beyond the process component borders, as provided interfaces (work products produced) or required interfaces (work products required).

Components flow execution is represented by process components' control interfaces. A control interface comprises a source component with an out port, a target component with an in port, and a connector establishing an association rule. The ports specify distinct interaction points between a component and its environment. Connectors are lines between the various ports, in order to express the connections that one wants to establish between process components (OMG, 2008). In this approach, the association rule related to the connector is represented by the control flow type: *sequence; fork; join; merge and decision*.

In this approach, a process component domain model is derived, being considered a modular view of the domain knowledge. This modular organization can improve the domain's comprehension, maintainability and, ultimately, its reusability. A mapping mechanism, described as heuristics, was defined as a guide to assist the mapping of properties from the feature model to the corresponding component model, guaranteeing their consistency and traceability. These heuristics were derived based on Blois et al. (2006), adapting the concepts of software products to software processes. Also, the elements defined in the metamodel and the process component model concepts were considered while defining the heuristics. Heuristics were proposed for mapping: (a) work unit features into components or internal elements of a component; (b) work products features into related interfaces or internal elements of a component; (c) roles into internal elements of a component; d) tools into internal elements of a component; e) control flows to control interfaces; and f) composition rules to restrictions. A more detailed explanation of this technique can be found in Teixeira (2016).

Figure 4 presents the component model of the Project Planning feature model (Figure 4) resulted after heuristics mapping application, as follows: (1) the activity features mapped into components, four of them are internal components of others; (2) all tasks were mapped to internal elements of components; (3) each relation between a work product and a work unit beyond the process component borders were mapped as an interface; (4) the Project manager role and the tool were mapped to components internal elements; (5) control flows were mapped to sequential control interfaces; and (6) composition rules to restrictions.

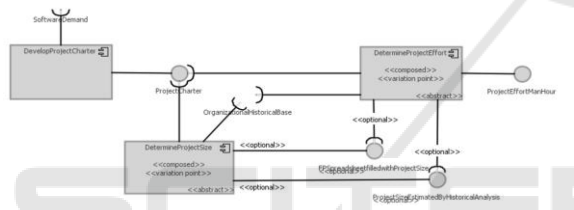


Figure 4: Project Planning SPrL Component Model (Adapted from Teixeira (2016)).

4 EVALUATIONS STUDIES

The approach has been evaluated and gradually refined through a series of studies that focused in different contributions. In this paper, SPrL representation and mapping heuristics studies are presented. It could be identified a higher expressiveness using the process variability notation proposed, where more semantic concepts inherent to SPrL scenarios can be described with more graphical representations. Also, the set of heuristics to perform mappings among artefacts was considered useful to keep the traceability of variability properties, relationships and restrictions.

An initial observational study was applied to evaluate the applicability of the first version of process domain metamodel and notation (Teixeira 2014). The same version was also used in an experimental study conducted in the context of a large oil and gas company in Brazil. Evolutions of the metamodel and notation were performed after these studies, including the use of cardinality, the component level development, specification of control and data flows, and treatment of variations in processes behaviours. Considering this last version, a more recent study was conducted (Teixeira, 2016), aiming to analyse the semantic and syntactic modelling capability and the expressiveness representation of reusable artefacts. A comparative analysis between the variability representation proposed and the one proposed in Barreto et al. (2011), was chosen as the assessment by the complexity involved in the analysed modelling notations that address semantically different concepts. Some results were: the proposed representation presented a higher expressiveness to represent more semantic concepts inherent to SPrL scenarios; and it has more graphical representations allowing more visual analysis of domain configuration points.

Another contribution evaluated was the feasibility of the mapping heuristics technique. An in vitro study was performed aiming to *analyse* a set of heuristics to generate a SPrL component model from a SPrL feature model in order to characterize, *with respect to* its effectiveness (*ratio between the number of correct component model elements created by a subject and the total amount of original component model elements*) and efficiency (*average time that the subject needed to create a correct component model element*) in generating SPrL component model *from the perspective of* Software Engineering researchers *in the context of* software developers (represented by five graduate students from a Software Reuse course) in one software process domain (see Figure 4).

As can be seen in Table 1, none of the subjects caught the total value of effectiveness (1). Although the subjects were not able to map all known elements from the feature model, all derived the expected components correctly, only presenting inversions in their internal structure type and internal variation type classifications. The major problems were related to work products relations and control flow mapping. A significant difference in the values of effectiveness and efficiency between the more and less experienced subjects was not observed. However, the time spent shows the complex involved in the activity.

Table 1: Execution Results.

Subj.	Exp. Level	Time (min)	#Correct Elements	Effectiveness	Efficiency
P1	0,57	128	34	0,68	0,271
P2	0,35	157	28,75	0,575	0,183
P3	0,25	81	23	0,46	0,284
P4	0,20	97	27,5	0,55	0,284
P5	0,19	90	21,5	0,43	0,239

The subjects filled an evaluation form after their mapping. Some subjects pointed out the need of some examples as useful additional information to support the heuristics application. No suggestion was indicated pointing out the need to reorganize the heuristics. Subjects did not identify any redundant heuristics. Most subjects agreed that a computational tool should support the mapping activity.

Some identified threats to the validity of this *quasi*-experiment are: the small sample size of the subjects; the absence of a comparison with another method; the academic environment, which is the same of the researchers. Although it was possible to get evidences of the technique feasibility, these issues indicate the need of further studies.

5 CONCLUSIONS

Software processes are complex dynamic systems, which involve several software engineering aspects and vary across projects. When this process family is managed by a systematic reuse approach, processes within an organization could be pro-actively organized, allowing for better tailoring to a specific project and organizational needs.

This paper presented a systematic process reuse approach. This approach combines SPrL and CBPD principles, also proposing a more complete variability process metamodel and notation, aiming to improve software process reuse, treating the variability aspect inherent to the process domain and considering modularity principles. The main contribution is related to supporting Domain Engineers in organizing knowledge from experienced process engineers and lessons learned in previous projects in a reusable way.

To complete SPDE, a set of criteria for components grouping were defined to support the organization of components derived in coarser-grained domain architectural elements, aiming to increase the domain understandability (Teixeira, 2016). Due to space limitation, it was not presented.

Also, a process reuse infrastructure was implemented, called Odyssey (Odyssey, 2017). This environment supports all phases of software reuse and was adapted to support SPrLs construction based on the proposed approach, including: (1) domain modelling in different abstraction levels described (feature and component models) and context models; and (2) mechanisms to support the application of the predefined heuristic mappings from features to components, an activity that is costly if done manually. Also, a verification mechanism was implemented to monitor the inconsistencies introduced during the modelling activity.

One of the limitations of this work is related to the preliminary support to the SPPE phase, which is being explored in further research. Case-based reasoning techniques are being studied to be applied in this scenario.

Further studies are planned to evaluate the whole approach in real scenarios or specific domains, aiming to validate the development of process lines in a real organization. It is important to involve software process experts.

ACKNOWLEDGEMENTS

This research was partially funded by CNPq.

REFERENCES

Alegría, J.A.H. and Bastarrica, M.C.: "Building software process lines with CASPER", In: Proceedings of International Conference on Software and System Process (ICSSP), Zurich, Suíça, IEEE (2012), 170 - 179.

Aoussat, F., Ahmed-Nacer, M., Oussalah, M. C.: New approach for software processes re-using based on software architectures. In: 15th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI'10), Orlando, United States, pp. 327-332 (2010).

Barreto, A., Murta, L., Rocha, A.R.: Software Process Definition: a Reuse-Based Approach. Journal of Universal Computer Science 17(13), 1765–1799 (2011).

Blois, A., de Oliveira, R., Maia, N., Werner and C., Becker, K. (2006) "Variability Modeling in a Component-based Domain Engineering Process", In: 9th International Conference on Software Reuse, Turin, Italy, June, Lecture Notes in Computer Science, Springer, Heidelberg, Germany, ISSN 0302-9743, pp. 395-398.

Chrissis, M.B., Konrad, M. and Shrum, S. (2006) "CMMI: Guidelines for Process Integra-tion and Product Improvement", In: 2nd ed., New York, USA: Addison-Wesley.

De Carvalho, D., Chagas, L. F., Lima, A. M., & Reis, C. A. L. (2014). Software Process Lines: A Systematic Literature Review. In Software Process Improvement and Capability Determination (pp. 118-130). Springer International Publishing.

Dias, Jaime W.; Oliveira Jr, Edson A. Modeling Variability in Software Process with EPF Composer and SMartySPEM: An Empirical Qualitative Study. In: ICEIS (1). 2016. p. 283-293.

Falbo, R. A., Bertollo, G., 2005, Establishing the Common Vocabulary for Software Organizations to Understand Software Processes. EDOC International Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE), Enschede, The Netherlands, 2005, pp. 1-8.

Fernandes, P., Werner, C., 2008, Ubifex: Modelling context aware software product lines. In 2nd International Workshop on Dynamic Software Product Line Conference, Limerick, Ireland, 2008, pp. 3-8.

Gary, K.A. and Lindquist, T.E., (1999) “Cooperating Process Components”, In COMPSAC99, pp.218-223. Phoenix, United States.

García-Borgoñon, L., Barcelona, M. A., García-García, J. A., Alba, M., & Escalona, M. J. (2014). Software process modeling languages: A systematic literature review. Information and Software Technology, 56(2), 103-116.

Ginsberg, M. P., and Quinn, L. H., 1995, "Process Tailoring and the Software Capability Maturity Model. Technica Report CMU/SEI-94-TR-024," Software Engineering Institute, Pittsburgh, PA1995.

Kellner, M.I.: “Connecting Reusable Software Process Elements and Components”. In 10th International Software Process Workshop, Dijon, France, pp. 8-11 (1996).

Kuhrmann, M., 2014, “You can't tailor what you haven't *modeled”. In: Proceedings of the 2014 International Conference on Software and System Process, pp. 189-190, ACM.

Kuhrmann, M., Méndez Fernández, D., and Ternité, T. 2016. On the use of variability operations in the V-Modell XT software process line. In Journal of Software: Evolution and Process, 28(4), 241-253. DOI: 10.1002/smr.1751.

Magdaleno, A. M., de Oliveira Barros, M., Werner, C. M. L., de Araujo, R. M., & Batista, C. F. A..: Collaboration optimization in software process composition. Journal of Systems and Software, 103, 452-466 (2015).

Martínez-Ruíz, T., García, F., Piattini, M.: Towards a SPEM v2.0 Extension to Define Process Lines Variability Mechanisms. In: Lee, R. (ed.) Software Engineering Research, Management and Applications. SCI, vol. 150, pp. 115–130. Springer, Heidelberg (2008)

Martínez-Ruíz, T., García, F., Piattini, M. and Münch, J.: "Modelling software process variability: an empirical study", In: Software, IET, 5, 2 (2011), 172,187.

Martínez-Ruíz, T., Münch, J., García, F. and Piattini, M.: “Requirements and constructors for tailoring software processes: a systematic literature review”. In: Software Quality Jour-nal, 20, 1 (2012), 229–260.

Northrop, L., 2002, “SEI’s Software Product Line Tenets”, IEEE Software, v.19, n.4, pp. 32-40, July/August.

Odyssey (2017) “Odyssey Project”, In: http://reuse.cos. ufrj.br/ odyssey

Oliveira Junior, E. A., Pazin, M. G., Gimenes, I. M., Kulesza, U., & Aleixo, F. A. (2013). SMartySPEM: A SPEM-Based Approach for Variability Management in Software Process Lines. In Product-Focused Software Process Improvement (pp. 169-183). Springer Berlin Heidelberg.

OMG, 2008: “Software Process Engineering Metamodel”, In: http://www.omg.org/technology/documents/ formal/spem.htm

Pillat, R. M., Oliveira, T. C., Alencar, P. S., Cowan, D. D., 2015, “BPMNt: A BPMN extension for specifying software process tailoring”, In: Information and Software Technology, 57, pp. 95-115.

Rombach, D., 2013, “Integrated Software Process and Product Lines”. In Perspectives on the Future of Software Engineering, pp. 359-366. Springer Berlin Heidelberg.

Sametinger, J., 1997, Software Engineering with Reusable Components, Springer-Verlag New York, Inc.

Schramm, J., Dohrmann, P., and Kuhrmann, M. 2015. Development of flexible software process lines with variability operations: a longitudinal case study. In Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. ACM (2015), 13. DOI: 10.1145/2745802.2745814.

Silvestre, L., Bastarrica, M. C., and Ochoa, S. F. 2014. A model-based tool for generating software process model tailoring transformations. In 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD). IEEE (2014), 533-540.

Softex, 2016, “MPS.BR - Brazilian Software Process Improvement, General Guide: 2016" [Online]. In: SOFTEX - Association for Promoting the Brazilian Software Excellence, Available: http://www.softex.br.

Teixeira, E.N., 2014, “A Component-Based Software Process Line Engineering with Variability Management in Multiple Perspectives”, In: 18th International Software Product Line Conference Doctoral Symposium, Italy.

Teixeira, E.; Mello, R.; Motta, R.; Werner, C.; Vasconcelos, A.. “Verification of Software Process Line Models: A Checklist-based Inspection Approach". In: XVIII Iberoamerican Conference on Software Engineering, 2015, Lima, p. 81-94.

Teixeira, E.N., “OdysseyProcessReuse: A Methodology for Component Based Software Process Line Engineering”, PhD Thesis. COPPE/UFRJ (In Portuguese), Rio de Janeiro, RJ, Brasil (2016).

Washizaki, H.: "Building Software Process Line Architectures from Bottom Up", In: Product-Focused Software Process Improvement (PROFES), Amsterdam, The Netherlands: LNCS, chapter 4034 (2006), 415-421.

Zakaria, N. A., Ibrahim, S., & Mahrin, M. N. R. (2015, August). The state of the art and issues in software process tailoring. In Software Engineering and Computer Systems (ICSECS), 2015 4th International Conference on (pp. 130-135). IEEE.