Software Process Commonality Analysis

Alexis Ocampo, Fabio Bella, and Jürgen Münch
Fraunhofer Institute for Experimental Software Engineering,
Sauerwiesen 6, 67661 Kaiserslautern, Germany
{ocampo, muench, bella}@iese.fraunhofer.de

Abstract

To remain viable and thrive, software organizations must rapidly adapt to frequent and often quite wide-ranging, changes to their operational context. These changes typically concern many factors, including: the nature of the organization's marketplace in general, its customers' demands, and its business needs. In today's most highly dynamic contexts, such as web services development, other changes create additional, severe challenges. Most critical are changes to the technology in which a software product is written or which the product has to control or use to provide its functionality. These product-support technology changes are frequently relatively 'small' and incremental. They are, therefore, often handled by relatively 'small,' incremental changes to the organization's software processes. However, the frequency of these changes is high, and their impact is elevated by time-to-market and requirements volatility demands. The net result is an extremely challenging need to create and manage a large number of customized process variants, collectively having more commonalities than differences and incorporating experience-based, proven 'best practices'. This paper describes a tool-based approach to coping with product-support technology changes. The approach utilizes established capabilities such as descriptive process modeling and the creation of reference models. It incorporates a new, innovative, tool-based capability to analyze commonalities and differences among processes. The paper includes an example-based initial evaluation of the approach in the domain of Wireless Internet Services as well as a discussion of its potentially broader application.

1. Introduction

Survival in today's highly dynamic business environments requires that organizations continuously adapt their processes. Success and growth — rather than mere survival — require that this adaptation be rapid enough to realize the competitive advantage offered by new business opportunities. Business models must be rapidly changed or newly developed; the organization's work force must be quickly updated and trained. Most challenging, however, is rapidly adjusting to changes in the organization's *process-support technology*. For organizations providing software products (*software organizations*), this includes the technology used to develop their products as well as the technology the products must control or use to provide their functionality. In many software-dependent arenas, for example Wireless Internet Services, process-support technology changes are individually small and incremental, but quite frequent. As a software organization adapts its development processes in these arenas, the result will inevitably be a large number of processes that vary in relatively minor ways. One way to control this proliferation and its attendant risks is to carry forward knowledge about what worked and what did not work in the past. In other words, software organizations working in this arena must be agile, but this agility should be based on prior experience rather than being merely hypothetical [1].

This raises the question: How can a software organization cope with product-support technology changes by rapidly creating customized software development processes containing proven 'best practices'?

The rationale for our work is a feeling that understanding an organization's current and past practices, describing the processes underlying these practices, and being able to identify variations and reasons for variations will certainly help software organizations address this question.

The basis for our approach is the creation of customizable, domain-specific process models (i.e., reference process models) through the bottom-up identification of process variations. The overall approach is described in [2]. In brief, its main steps are:

Set-up pilots - Suitable pilot projects are determined and organized.

Perform pilots - The pilot projects are conducted.

Observe and model processes - The processes as performed in the pilot projects are observed and modeled.

Identify and evaluate processes and practices from

related fields - This information will be used to complete the reference process model where it is incomplete.

Analyze commonalities and differences - Commonalities and differences between the different process models are analyzed in order to identify process variants and justifications for them. This must recognize differences in the application domain as well as goals and contexts of the pilot projects.

Create comprehensive process model - The models for the processes used in the pilot projects, as well as practices and processes from related fields, are integrated to create a comprehensive process model [3]. In this paper, we call the resulting comprehensive process model a *reference process model*, because it is intended to be used as a reference for developers and managers that provides a starting point for developing a customized process meeting the requirements for a set of product-support technologies.

The tool-supported technique presented in this paper supports the activity *analyze commonalities and differences* and can be helpful in practical situations where software organizations must compare a set of process models in a systematic way, in order to understand their context-dependant variations.

The following sections discuss the background for our work, describe the details of the technique we have developed, provide a preliminary evaluation of its value (in terms of an example of its use), and discuss possible future work.

2. Background

The following section presents commonality analysis performed previously in related fields as well as a description of the context of the work.

2.1. Related Work

In the database world the problem of integrating schemas of existing databases from the perspectives of different users (database schema integration) is addressed by [4]. Products from this database integration are: a global database schema, data mapping from global to local databases, and mapping of querying transactions from local to global databases. Semantic relationships between database schema X1 and database schema X2 are defined as: identical, equivalent, compatible, and incompatible. The schemas are analyzed and compared in order to uncover conflicts. Any situation where the representations of X1 and X2 are not identical is considered to be a conflict between X1 and X2. The representations of the schemas are used to compare them, but there is no defined method to do this comparison.

Integration of design specifications has been examined by [5], [6], [7]. These approaches have in common that they integrate pairs of specifications and use specification formalisms, and that their goal is to reduce the complexity of the global specification. The analyst compares components of both specifications and declares them equivalent or not. A special formalism is used in order to conclude when a component X1 is equivalent to component X2. Conflicts are uncovered when ambiguities and inconsistencies are detected between pairs of specifications. Negotiations are needed between developers in order to identify and resolve conflicts. Once the integration has been accomplished, there is no way to extract the original views from the final specification, which is not the case with the technique presented in this article.

In the product line world, identifying commonalities and differences is an accepted, wide-spread practice when comparing systems [8]. Usually, common elements are reused and variations are hidden, in the most appropriate way, in order to produce a family of products. In order to understand the extent of commonality and variability in a family of products, the proposed steps are:

Establish the scope - The collection of objects under consideration.

Identify commonalities and variations - Similar attribute values across the family members are identified. Variants of the attribute values are identified. The attribute values justify the variants as the differences of compared processes are justified by their context.

Bound variations - A range of values for the variants is defined.

Exploit commonalities and accommodate variations - The results of the commonality analysis are grouped into procedures, inheritance, and parametric polymorphism.

In the process modeling world, there exist some approaches to integrating partial process models (e.g., views) into a descriptive process model when persons covering different roles describe their perspectives of a large software process [9], [10], [14]. In these approaches, variations are often seen as inconsistencies or as imprecision, and therefore, trigger questions that lead to a review of the process that will eliminate these inconsistencies. The final goal is to obtain a multi-view-consistent comprehensive process model. In our approach, some of the rules discussed in [14] are adapted to the SPEARMINT environment [11] and applied in order to create a reference model with common best practices and variations.

In the process modeling world, introducing reference models is often done in a top-down fashion using prescriptive process models. Prescriptive process models describe how a product *should* be developed. Typically, prescriptive process models lack tailoring guidelines, are generic (i.e., do not define specific approaches to carrying out activities), and do not describe a company's actual processes.

The commonality analysis technique we propose can

be seen as analogous to the commonality analysis of products in product line approaches. It relies on

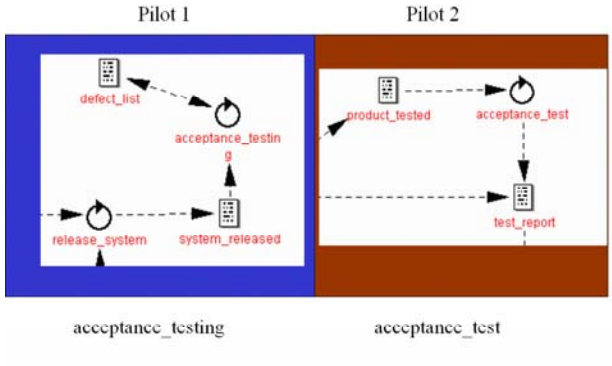

Figure 1. Manual commonality analysis (excerpt)

descriptive, rather than prescriptive, process modeling to create a reference model, utilizing several capabilities found in multi-view modeling approaches (such as rule-based comparisons).

2.2. Context

The technique was developed and evaluated as part of the WISE project. The project aims at producing integrated methods and components (COTS and open source) to engineer services on the wireless Internet. The components include a service management component and an agent-based negotiation component. Two pilot services, i.e., a financial information service and multi-player game, are being developed by different organizations. The project lasts 30 months and an iterative, incremental development style is applied: three iterations are performed of roughly 9 months each. In iteration 1, a first version of the planned pilot services was built using GPRS. At the same time, a first version of methods and tools was developed. In iteration 2, a richer second version of the pilots was developed on GPRS, using the first version of methods and tools. In parallel, an improved second version of methods and tools was developed. In iteration 3, the final version of the pilots is being developed on UMTS, using methods and tools from the second iteration. Also, a final version of methods and tools is being developed. One of WISE's objectives is to develop a reference process model that may be used by software organizations for creating wireless Internet services. In order to achieve this objective, it was decided to use the described empirical approach, i.e., the observation of realistic pilot projects, techniques, and processes.

3. Technique

The proposed commonality analysis technique can be performed solely manually, or with the use of a specifically developed tool, SPEARSIM.

3.1. Manual Commonality Analysis

In order to perform a commonality analysis, the models must be rigorous. This may be achieved, for example, by using electronic process guide (EPG) capabilities with graphical views [11]. Once similar process parts are identified, a process engineer reads the definition of the processes and products related to these parts. After reading and analyzing the descriptions, the process engineer makes an assumption that two or more processes or sub-processes are similar or different. For example, Figure 1 shows two example process models. It can be seen that the activities *acceptance_test* and *acceptance_testing* are similar.

The process engineer has to check the descriptions of processes, products, roles, and tools in order to establish an assumption that they are similar. The next step is to check the assumption by reviewing the identified commonalities with the process performers, that is, the observed developers, in order to obtain a common agreement on the commonalities. If the activities are not similar, then the next step is to find possible reasons for the variation. The reasons can usually be found in the context of the process. A characterization vector describes the environment in which the process model was elicited. A characterization vectors excerpt is shown in Table 1.

Table 1. Characterization vectors

Customization factor	Characteristic	Project 1	Project 2
Domain characteristics	Application type	Information system	Computation intensive system
	Business area	Mobile online trading services	Mobile online entertainment services
Development characteristics	Project type	Client System adaptation	Client New development Server New development
	Transport protocol	GSM/GPRS/UMTS	GSM/GPRS/UMTS
	Implementation language	WML	Client: J2ME Server: J2EE
Enterprise characteristics	Organizational context	Investnet-Italy	Motorola MTCI
	Role	Service provider, content provider, service developer	Technology provider, service developer

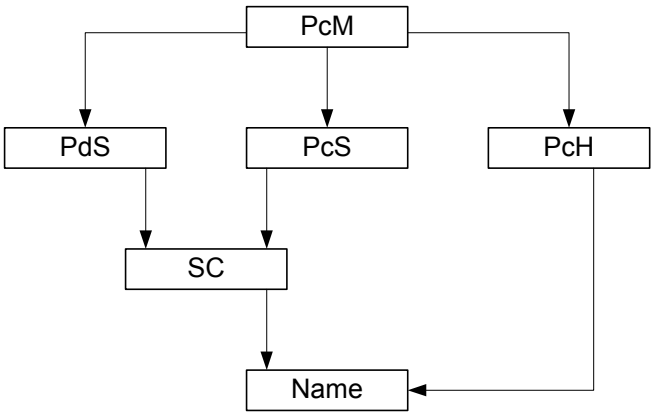

Figure 2. Rules overview.

3.2. Tool-supported Commonality Analysis

The tool SPEARSIM is designed to support a process engineer in comparing large and complex processes. The rules differ in their degree of complexity: on the one hand, simple rules can be used to compare entity names (such as process or product identifiers) and help to identify synonyms and homonyms; on the other hand, more complex rules can be used to compare aggregation structure of products and processes. The dependencies, represented by arrows, show that the computation of complex similarities rests upon data computed by simpler rules. Figure 2 shows an overview of the rules defined and their dependencies. The results of some rules are presented to the process engineer as assumptions. In order to focus certain aspects of the models assumed to be relevant according to the given context, the process engineer is able to influence the importance of the different rules by setting parameters (weights).

In the following, the single rules are discussed in more detail:

PcM (Process Model) - The similarity values are computed by the PcM rule by building a weighted sum of the rules: PdS, PcS, and PcH.

PdS (Product Structure) – The similarities between two processes are computed by the PdS rule resting upon the homogeneity of the sets of products the two processes access, i.e., the products they produce, consume, or modify. The PdS rule applies the SC rule, which is discussed below.

PcH (Process Hierarchy) - The PcH rule computes similarities between processes by analyzing the hierarchy of their sub-processes. Since a comparison of the entire aggregation trees can become very complex, the computation is only concerned with the first three hierarchy levels of the tree structure. The PcH also involves a direct analysis of the similarity of the sub-processes, which is performed by the Name rule discussed below.

PcS (Process Structure) - The PcS computes similarity assumptions between two processes resting upon the homogeneity of the sets of sub-processes they aggregate. The PcS rule, like the PdS, applies the SC rule.

SC (Structure Compatibility) - The SC rule can be applied on two sets of processes or products. The value computed by SC represents the degree of homogeneity of the two sets, i.e., how well the entities of one set match the entities of the other set. For example, given two sets $A = \{a, b, c\}$ and $B = \{d, e, f\}$ where b and f are the only identical entities between the two sets, i.e., the number of matches is $m = 1$ and the maximal number of matches is $n = 3$, the similarity value returned by SC is computed as:

$$\frac{m}{n} = \frac{1}{3}$$

Name - This rule computes text similarity according to the Levenshtein distance [12]. The Levenshtein distance (LD) is a measure of the similarity between two strings, which we will refer to as the source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform s into t. For example, If s is "test" and t is "test", then LD(s, t) = 0, because no transformations are needed. The strings are already identical. If s is "test" and t is "tent", then LD(s,t) = 1, because one substitution (change "s" to "n") is sufficient to transform s into t. The greater the Levenshtein distance, the more different the strings are. This rule delivers the basis for the whole computation at the beginning of the analysis process, i.e., when the process engineer has not set facts yet.

The tool analyzes the similarity of two process models using the above clarified rules, which formalize different similarity aspects that may occur between entities of two process models. The process engineer converts assumptions into facts by accepting or rejecting the assumptions developed by the tool. The process engineer may need to read the descriptions of the compared processes, artifacts, roles, or tools, in order to have an adequate basis upon which to accept or reject the assumptions. Once the facts are established, the tool uses them to re-analyze the two models and present a new set of assumptions to the process engineer, who decides whether to continue with a new iteration by establishing new facts or whether to stop the comparison. Figure 3 shows an excerpt of the table of commonalities: In this case, the process engineer has turned all the assumptions into facts.

The degree of dependency of the different rules can be changed by setting parameters (weights): In order to achieve a sharper picture, the process engineer can trim the weights and try to get most of the greatest similarities computed for the pairs expected to be identical, most of the lowest similarities for the pairs expected to be completely different and, at the same time, to maximize

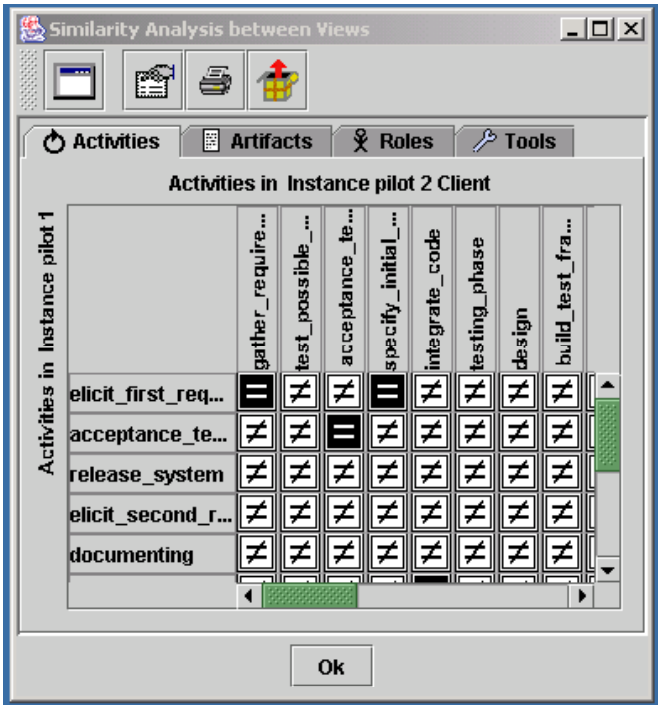

Figure 3. Commonalities table (excerpt)

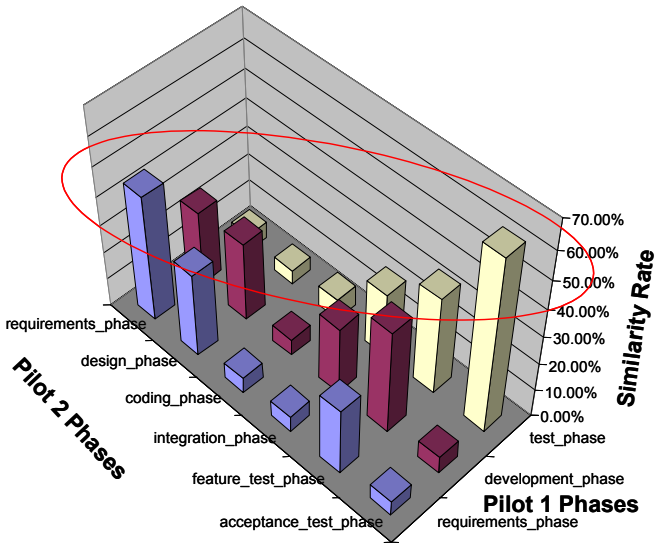

Figure 4. Commonalities view generated by SPEARSIM.

the difference between great and low similarities.

The resulting table of commonalities can be used in reviews with developers to obtain a common agreement. Expected great similarity values indicate the common path between the compared processes. Unexpected similarity values characterize the most interesting pairs of process entities, since they could indicate variations between the compared processes.

Comparing more than two process models requires a sequence of binary comparisons. Ladder strategies, as well as other techniques, have proven useful for identifying commonalities across a set of processes.

4. Evaluation

Within the WISE project, two process models with 11 and 13 sub-processes, respectively, were compared. In a first step, a manual comparison was performed and pairs of similar process parts were documented as shown in the examples appearing in Figure 1. In a second step, the SPEARSIM tool was used. Similarity facts between products were established by the process engineer according to the content and purpose of the documents manipulated by the different processes.

Since neither tool assets nor roles were modeled at this stage of the project, rules referring to these entities were ignored. In a third step, a computation was performed in order to analyze commonalities among the processes within each phase. Finally, another computation was performed in order to analyze commonalities between the different phases of the two development processes.

Figure 4 presents a view, generated by SPEARSIM, showing the similarities between phases.

The analysis of the commonalities among phases was performed with PcM = PdS · 0.609 + PcH · 0.138 + PcS · 0.253. In this case, the weights were set for taking into account the structures of the different phases (PcS), i.e., which processes they aggregate directly, as well as the hierarchy of the constituent processes (PcH). The structure of the products the phases access was considered, too (PdS).

The phases of both models were settled in a chronological order in the diagram. As a consequence, the greatest similarities could be expected along the main diagonal (highlighted by the ellipse). As mentioned above, the part of the diagram not matching the expectation may indicate variations in the two processes or evaluations by the tool that were too optimistic. The diagram shows the greatest commonalities in the requirements as well as in the test phases of the two development processes. These results were also observed in the manual analysis, an example of which can be seen in Figure 1, where basic activities of the testing phase were declared similar. A mismatch of the development phase (pilot 1) and the coding phase (pilot 2) shows where to expect the greatest differences between the two development processes. The main reasons for the differences were found in the different maturities of the software development organizations responsible for the development of the pilot services as well as in the different final products, a WML-based information system in the case of pilot project 1, and a distributed

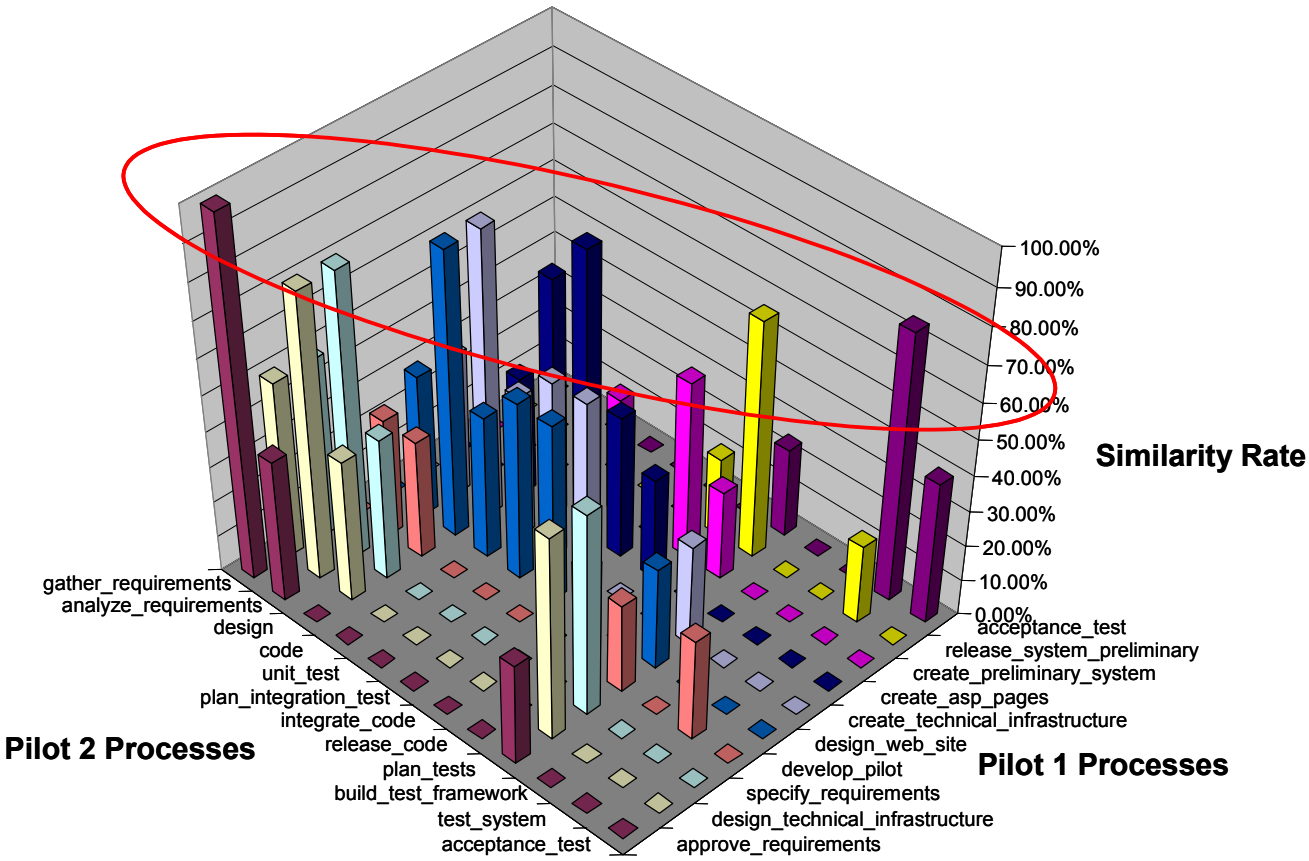

Figure 5. Commonalities view generated by SPEARSIM.

game implemented in Java in the case of pilot project 2. The great similarities between the requirements and the test phases of the two processes, respectively, indicate an example of an optimistic similarity computation due to the underlying similarity of the products manipulated by these phases (i.e., products concerning requirements). Although these are obvious results, the objective of this analysis goes further, because we intend to uncover process similarities across different projects (i.e., final products) in a domain (e.g., wireless Internet Services) which we can then nominate as 'best practices' that should appear in any customized version of the reference process. Therefore, going back to the results of both analyses, our proposal will then be that new projects intended to develop a software product in the wireless Internet services domain should include a subprocess with the scope and characteristics of the acceptance test as part of their process.

Figure 5 shows the similarities computed between the underlying processes, which are arranged on the axes in a chronological order. The computation was performed with PcM = PdS · 1.0. The weights were chosen in order to consider only the structure of the products accessed by the constituent processes: As the processes were almost not aggregated or the aggregations were not comparable further, an analysis of their structures was avoided. Although a more complex situation is given here, in this case, most of the greatest similarity values are also arranged, as expected, along the main diagonal of the diagram. The reasons for some exceptions can be found in different granularity degrees of the models as well as in different experience levels of the two software development organizations, particularly with respect to testing.

The results produced were useful for the creation of the reference process model. Figure 6 shows an excerpt of the reference process model.

This excerpt taken from the WISE project shows merged experience-based, proven 'best practices' and variations. The process engineer merged, for example, the 'best practices' *approve requirements* and *gather requirements*, as one part of the common process named *gather requirements*. The same was done with the processes *specify requirements* and *analyze requirements* naming a part of the process *specify requirements*. On the other hand, pilot 1 did not present evidence of performing the activity *build test framework*, therefore, it was considered different by the process engineer and process

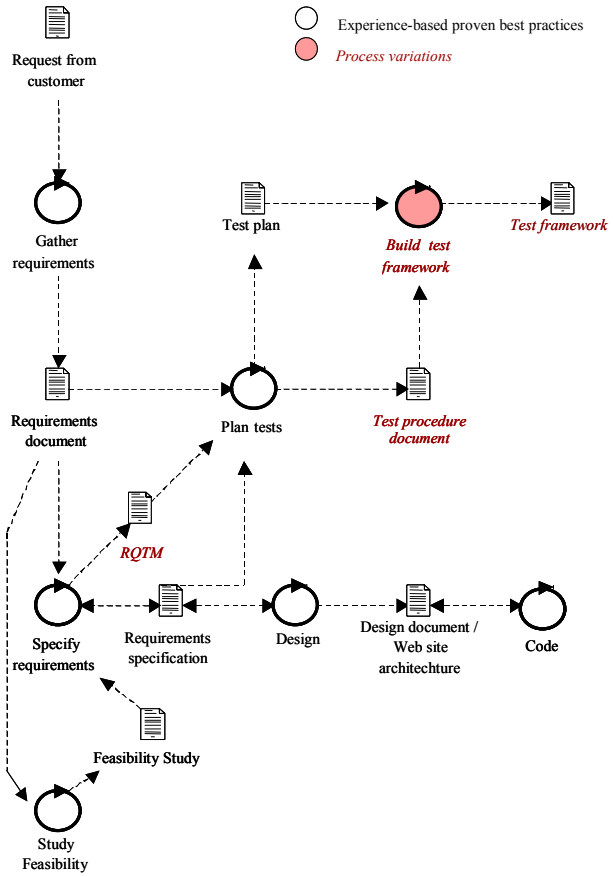

Figure 6. Reference process model (excerpt).

owners and merged into the comprehensive process model as an possible variation of the process.

5. Summary and Outlook

The tool presented here is helpful for managing the comparison of large, complex processes, and the rules are applicable for processes in the same organization as well as in different organizations. Nevertheless, in the course of the exercise discussed in the previous section, we noticed that some assumptions made by the tool were not as concise as we expected. This suggests that it would be good to perform further research on similarity computation rules, in particular, rules followed intuitively by process engineers, in real practice, to determine best practices and process variations. Also, some additional process attributes (e.g., measures, estimates, standards applicable, pre-post-conditions, etc.) are not considered in the actual version of the tool. Future research may then address the following questions: What are appropriate rules to determine process similarity and dissimilarity? When should these rules be applied, and when not? Which degree of process complexity requires automated similarity analysis?

Capabilities for identifying best practices and process variations might be valuable for software organizations in other areas too, such as:

- Process measurement: Metrics that reflect process similarities and differences could be important for guiding process improvement.
- Process training: Identifying the gaps between desired and actual processes could improve workforce training.
- Tailoring guidance: Notations for describing common and alternative process parts could support tailoring.
- Outsourcing: Commonality analysis could provide a basis for integrating processes between an outsourcing organization and its outsourced-to organizations.
- Executive decision making: Notations for assigning value to variations can be important for managers of software organizations for deciding how to change a software project so that it may proceed more effectively and efficiently.

Acknowledgments

This work has been funded by the European Commission in the context of the WISE project (No. IST-2000-30028). We would like to thank the WISE consortium, especially the pilot partners, for many fruitful and interesting interactions. We would also like to thank Dr. William E. Riddle for his valuable comments on the article, and Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first version of the article.

References

[1] Boehm, B.: Get Ready for Agile Methods, with care. IEEE Computer, vol.35, pp. 64-69, Jan 2002.

[2] Becker-Kornstaedt, U., Boggio, D., Münch, J., Ocampo, A., Palladino, G.: Empirically Driven Design of Software Development Processes for Wireless Internet Services. Proceedings of the Fourth International Conference on Product-Focused Software Processes Improvement PROFES, 2002.

[3] Ocampo, A., Boggio, D., Münch, J., Palladino, G.: Toward a Reference Process for Developing Wireless Internet Services. IEEE Transactions on Software Engineering, vol. 29 No. 12, pp. 1122-1134, Dec 2003.

[4] Batini, C, Ceri, S, Navathe, S.B.: Conceptual Data Base Design: An Entity-Relationship Approach. Benjamin Cummings. Redwood City. (1992).

[5] Feather, M.S.: Detecting Interference when Merging Specification Evolutions. Proceedings of the Fifth International Workshop on Software Specification and Design, pp. 169-176 (1989).

[6] Leite, J.S.P., Freeman, P.A.: Requirements Validation Through Viewpoint Resolution. IEEE Transactions on Software Engineering, vol. 17, No. 12, (1991).
[7] Robinson, W.N.: Integrating Multiple Specifications Using Domain Goals. Proceedings of the Fifth International Workshop on Software Specification and Design, pp. 219-226 (1989).
[8] Coplien, J., Hoffman, D., Weiss, D.: Commonality and Variability in Software Engineering. IEEE Software, vol. 15, No. 16, pp. 37-45, Dec 1999.
[9] Turgeon, J., Madhavhji, H., N.: A Systematic, View-Based Approach to Eliciting Process Models. In proceedings European Workshop on Software Process Technology, pp. 276-282, 1996.
[10] Verlage, M.: An Approach for Capturing Large Software Development Processes by Integration of Views Modeled Independently. Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering SEKE, 1998.
[11] Becker-Kornstaedt, U., Scott, L., Zettel, J.: Process Engineering with Spearmint/EPG. Proceedings of the 22nd International Conference on Software Engineering ICSE, 2000.
[12] Levenshtein V, I.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady 10(8) 707-710, (1966).