Software Process Commonality Analysis

Authors:
Alexis Ocampo
Jürgen Münch
Fabio Bella

Submitted to ICSE 2004

IESE-Report No. 090.03/E
Version 1.0
September 2003

A publication by Fraunhofer IESE
Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft. The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
67661 Kaiserslautern
Abstract

One of the major challenges in software development projects consists of defining appropriate development processes and tailoring them to project and organizational constraints. One solution for this problem is to develop domain-specific adaptable reference process models based on empirically gained experience, observation of real processes, and comparison of these processes. This article presents a technique for analyzing commonalities and differences between process instances and shows how to use this knowledge for creating software process reference models. The technique is based on different comparison strategies, which can be performed manually or tool-based (with the specifically developed tool SPEARSiM). The commonality analysis technique was evaluated in a case study in the wireless Internet service-engineering domain. Evaluation results are used to illustrate the technique.
# Table of Contents

1 Introduction 1  

2 The commonality analysis technique 4  
2.1 Manual comparison 5  
2.2 Rule-based semiautomatic comparison 6  
2.3 Reference process model creation 9  

3 Validation 12  
3.1 Case study 12  
3.2 Results 13  
3.3 Lessons learned 17  

4 Related work 19  

5 Summary and outlook 21  

Acknowledgments 22  

References 23
Emerging technologies often provide software development organizations with new and different ways to develop and deploy software for staying or becoming competitive. However, managers and developers often find themselves immersed in a sea of technology that could be of use, but without having a clear picture on how to proceed [20]. Usually, for new and therefore unknown application domains, no explicitly defined software development processes are available, and it is unclear how to adapt existing practices to those domains. Furthermore, introduction of new processes, or the modification of existing processes, is very risky, because typically there exists no previous experience on the effects of the changes in the environment of the software development organization [15]. Constraints such as time to market and competitiveness often make it difficult or nearly impossible for software development organizations to put their efforts into designing and carefully implementing appropriate process models.

As a result, organizations often lack processes for adapting to context changes, and somehow, what used to be valid for past projects, is not valid anymore.

Context changes (e.g., changes due to technology, to organizational policies, or to new market demands) lead software project managers and developers to questions such as: Which development techniques, methods, and tools are appropriate, and how can I combine them? Which development steps have to be performed? How to tailor development processes to my new context? It would be nice to have a software process reference model at hand. This is because one major objective of a software process reference model is to provide guidance for developing software products in a given context.

We use the term software process reference model for a process model that integrates consistent and validated empirical and theoretical evidence of processes, products, roles and tools used for developing software in a domain. Additionally, such a model should provide tailoring guidelines. Tailoring guidelines describe relationships between variants of the software process reference model and the context of its application in a specific organization. The question that arises is: How to define such a software process reference model?

Figure 1 shows an existing method that has been defined for empirically designing a reference process for new application domains [3]. The method is based on a bottom-up approach, where knowledge is elicited from organizations and formalized through incremental cycles. The method uses both the experience captured from the execution of pilot projects [19] (considered to be
representative projects of the domain after detailed market studies) and collected experience found in literature or industry reports. Descriptive process modeling is used to capture the information of the developers as they execute their tasks. A literature search is used to capture experience from related contexts. Once all this information has been collected, it is integrated in a systematic way in order to end up with a reference process model suitable to the application domain. In order to accomplish that objective, the method proposes two activities: first, an analysis of commonalities and differences between the involved process models, in order to identify process variants and special characteristics of the application domain that create such variants, and second the creation of a reference process model with common, optional, and alternative parts.

Figure 1. Method for designing a reference process [3].
This paper focuses on the commonality analysis activity (shadowed in Figure 1) and presents a new technique for identifying commonalities and differences between processes. The technique is based on two complementary procedures: A manual procedure, where a process engineer performs a manual comparison based on his knowledge and observations from projects and their contexts, and a rule-based semiautomatic procedure, where a process engineer with the support of a specifically developed new tool, SPEARSIM, compares descriptive process models.

The paper is ordered as follows: Section 2 describes the steps of the commonality analysis technique and the underlying principles of the commonality analysis tool and its features, and presents how to create the reference process model. The validation of the commonality analysis technique is described in Section 3: The application of the technique in a case study is surveyed, the results from the manual and semiautomatic analysis are presented, and a discussion on the advantages and disadvantages of both types of analysis is shown. Section 4 surveys related work, and finally, Section 5 summarizes the article and gives an outlook on future work.
2 The commonality analysis technique

A commonality analysis helps a process engineer to uncover similarities and differences between two or more process models. Once the commonalities and differences are clarified, the process engineer is expected to have a clearer view and can identify process variations.

![Diagram](image)

Figure 2. Analysis of commonalities and differences.

Figure 2 shows the artifacts, roles and tools involved in the analysis of commonalities and differences. We proposed to use prototype projects (so-called pilot projects) for new domains in order to have the creation of the reference model on empirical evidence. Pilot projects should be carefully selected so that their context is similar to the expected context of projects intending to use the reference process model in the future [17].

The pilot’s requirements and their contexts have to be defined before the commonality analysis (within the activity set-up pilots; see Figure 1). Usually the requirements that must be fulfilled by the pilots are the result of a marketing study, which tries to predict future successful business cases, or the result of an abrupt context change, which demands a new process model. The context that surrounds the pilots must be defined in order to make a valid analysis of com-
The commonality analysis technique

monalities. One way to describe the context is through a characterization vector. A characterization vector describes the context with \(<\text{characteristic, value}>\) pairs [1]. According to [5], the characteristics can be identified using typical questions like: What kind of product is being developed? What is the experience of the organization? What are the implementation languages?

In the commonality analysis, the process engineer who usually is responsible for eliciting process knowledge from process owners, and capturing this information into a descriptive process model [4], is responsible for comparing the entities of those models, interviewing the process owners, and making assumptions about the similarities or differences between them. The process owners are responsible for providing detailed information to the process engineer on the descriptive process models. Developers or project leaders can be process owners.

2.1 Manual comparison

The following are the suggested activities for performing a manual commonality analysis:

1. **Identify strategy:** The strategy defines how to go through the process models and compare them systematically. Using as a basis the modeling strategies discussed in [11], we defined the following strategies for performing the analysis: phase-based, level-based, and concept-based. Phase-based is defined as going through the process models’ phase by phase from start to end. Level-based is oriented towards different aggregation levels of the process models. The models are followed in a top-down manner. Concept-based is defined as comparing the elementary products descriptions, the product hierarchies, the elementary process descriptions, and the process hierarchies. The strategy needs to be selected individually for a project. For example, if there is a good understanding of the control flow a phase-based strategy might be the most appropriate. Besides it is also possible to combine strategies.

2. **Compare entities:** Once the strategy is identified, the process engineer performs a comparison between the processes. By using, for instance, electronic process guides (EPGs) [13], the process engineer can compare process models and identify similar process parts.

Once a similar process part is identified, the process engineer reads the definition of the processes and products in the area. After reading and analyzing the descriptions, the process engineer could affirm that two or more processes or sub-processes are similar or different. Previous experience of the process engineer with the descriptive process models and his knowledge of the context influence this affirmation. The process engineer records the information of similar and different processes and products with three simple column tables (see Table 1). A “+” value is given to processes/products that seem to be similar. A “–
The commonality analysis technique

" value is given to a process/product of Pilot X that seems to be different from all of the processes/products of Pilot Y. The symbol "*" is used to represent all the processes/products of a given process model. A "?" value is given to processes/products that are alike, but not similar, i.e., their purpose is the same, but the steps suggested by the process are different. Those are possible candidates for alternative parts in the reference process model.

2.2 Rule-based semiautomatic comparison

The idea of semiautomatic comparison can be explained as a loop consisting of four steps:

<table>
<thead>
<tr>
<th>Pilot X</th>
<th>Pilot Y</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>procX1</td>
<td>procY1</td>
<td>+</td>
</tr>
<tr>
<td>procX2</td>
<td>procY2</td>
<td>-</td>
</tr>
<tr>
<td>*</td>
<td>procY3</td>
<td>-</td>
</tr>
<tr>
<td>ProcX4</td>
<td>*</td>
<td>-</td>
</tr>
<tr>
<td>Products</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>prodX1</td>
<td>prodY1</td>
<td>+</td>
</tr>
<tr>
<td>prodX2</td>
<td>prodY2</td>
<td>+</td>
</tr>
<tr>
<td>prodX3</td>
<td>prodY3</td>
<td>?</td>
</tr>
</tbody>
</table>

Table 1. Commonalities and differences.

**Step 1:** A tool should propose which parts of two development process models are similar by providing assumptions. The computation of such assumptions would rest upon rules, which formalize different similarity aspects that may occur between entities of two process models [14].

**Step 2:** The process engineer should turn some assumptions into facts, i.e., accept or reject some of the assumptions provided by the tool.

**Step 3:** The facts should be used by the tool in order to perform an improved computation of the residual assumptions.

**Step 4:** Finally, the tool should present the results of the computation to the process engineer, who has to decide whether to iterate the loop by setting new facts and initiating a new computation or whether to stop the comparison.

The integration of a rule-based comparison approach into the SPEARMINT environment [4] enables the definition of complex rules by means of the conceptual richness offered by the language defined by this environment.
Figure 3 shows an overview of the rules defined and their dependencies. The rules differ in their degree of complexity: on the one hand, simple rules can provide similarity data about entity names, see the rule Name depicted at the bottom of the figure; on the other hand, more complex rules can handle information like the aggregation structure of products and processes, e.g., PdS, PcS, and PcH depicted on top of the figure. The dependencies, represented by arrows, show that the computation of complex similarities rests upon data computed by simpler rules. The results of the PcM rule 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 should be able to influence the degree of dependency 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.
The commonality analysis technique

**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 \( m = 1 \) and the maximal number of matches is \( n = 3 \), then 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 [21].

SPEAR SIM implements the rule-based comparison approach discussed in this section. The tool has become an integral part of SPEARMINT.

Figure 4 shows the user interface of the tool. The square icons depicted in the figure represent assumptions as proposed by the system and facts as set by the process engineer. A filled square \( \square \) indicates great similarity between the two entities on the x and on the y axis, respectively. An empty square \( \square \) means no commonalities. An icon like \( \square \) means moderate similarity. Facts are represented by the icons \( \square \), for different entities, and \( \square \), for identical entities.
2.3 Reference process model creation

The section presents how the results from the commonality analysis are used in order to integrate the process models, and to derive guidelines for future use of the software process reference model. Figure 5 presents the process model. The characterization vector, the commonalities and differences, and the pilots’ requirements resulting from the previous process are used here. The following activities are proposed:

1. The process engineer selects pairs of processes whose value is “=” in the table of commonalities and differences. For each pair, the process engineer asks the process owners whether they believe that the processes have the same goal. If this is the case, a new name and description is written. Next, the process engineer pastes the new common process into the reference process, creating what can be called a common part of the reference process model. If the
process owners do not believe that the processes are common, then another value is given to the compared pair of processes.

2. The process engineer selects tuples of the form (*, procY, -) or (procX, *, -) from the table of commonalities and differences. These are the cases of a process that exists in only one process model, or that is different to all of the processes of the compared process model. For each tuple, the process engineer asks the process owners whether they believe that the unique process can be added in the reference process model, or whether that is not possible. This question is justified by the fact that the process owner, who does not follow the mentioned process, could see it as an opportunity for improvement. If this is the case, a new name and description is written, and a new common process is included in the reference model. On the other hand, the process owner could have no interest or no resources for following the mentioned process. In this case, the process is added as an optional part in the reference process model. Optional means that the process could or could not be instantiated by process owners.

3. The process engineer selects pairs whose value is “?” in the table of commonalities and differences. This occurs with processes whose purpose is similar, but whose procedures are different. For each pair, the process engineer discusses the following with the process owners:
Could both process owners enact both processes? If this is the case, then a new question arises: Which process is more suitable, adequate and useful for both process owners? At the end of the discussion, an agreement on one process
should be obtained. A new name and description is given and the new common process is included in the reference process model. If both process owners could not enact both processes, then they are declared alternative, and included in the reference process model. Alternative processes are those that are similar in purpose but different in procedure.

4. Identify gaps and fill them with best practices found in a literature search: After merging the process models, gaps are expected, especially when the domain is not well understood, and none of the process models contain practices or techniques to solve new challenges.

5. Derive tailoring guidelines: The created model should provide means to identify which project’s characteristics forces a process owner to enact a given alternative or to enact an optional process. By using the characterization vectors, the process engineer and the process owners identify the characteristics, which justify the common, optional, and alternative parts of the comprehensive process model. How to formalize these tailoring guidelines? This is a question of our current research. There are formal methods such as decision trees that could be useful. On the other hand, an informal, clear description can be enough.
3 Validation

The technique was evaluated by a “slice of life” example [18] in a case study comprising two pilot projects. The case study to be described was conducted in the context of the WISE project (Wireless Internet Services Engineering). The overall goal of the project is the development of a methodology and architecture for developing wireless Internet services.

3.1 Case study

In the WISE project, pilots are a means for designing processes and understanding the technology and methodology to engineer and operate with Wireless Internet Services in realistic contexts and different application domains. Based on market demands (such as the need to adapt existing services for the Internet towards Wireless Internet Services or to create new services) and companies’ interests, two target contexts were defined: the development of a Wireless Internet service for mobile online trading and the development of a service for Mobile Entertainment.

Industrial partners responsible for the pilot development and the underlying infrastructure are Motorola Global Software Group - Italy (Motorola GSG-Italy), and Investnet (Italy). As success factors for wireless Internet services, the industrial partners identified time-to-market, and the ability to quickly deliver functionality with simultaneous fulfillment of high quality requirements and high usability requirements in terms of service performance.

Table 2 presents the characterization vectors used in the context of the WISE project.
### Characterization vectors

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Pilot 1</th>
<th>Pilot 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application type</td>
<td>Information system</td>
<td>Computation intensive system</td>
</tr>
<tr>
<td>Business area</td>
<td>Mobile online trading services</td>
<td>Mobile online entertainment services</td>
</tr>
<tr>
<td>Project type</td>
<td>Client - System adaptation</td>
<td>Client - New development</td>
</tr>
<tr>
<td></td>
<td>Server - New development</td>
<td>Server - New development</td>
</tr>
<tr>
<td>Requirements technique</td>
<td>Structured text / UML cases. Intended screens, masks,</td>
<td>Structured text / UML cases.</td>
</tr>
<tr>
<td></td>
<td>forms and outputs</td>
<td></td>
</tr>
<tr>
<td>Transfer protocol</td>
<td>GSM / GPRS / UMTS</td>
<td>GSM / GPRS / UMTS</td>
</tr>
<tr>
<td>Implementation language</td>
<td>WML</td>
<td>Client: J2ME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Server: J2EE</td>
</tr>
<tr>
<td>Validation technique</td>
<td>Black box testing</td>
<td>White box unit testing, feature testing on target terminal</td>
</tr>
<tr>
<td>Business objectives</td>
<td>Capturing knowledge, reducing time to market, high quality products</td>
<td>Capturing knowledge, reducing time to market, high quality products</td>
</tr>
<tr>
<td>Role</td>
<td>Service provider, content provider, service developer</td>
<td>Technology provider, service developer</td>
</tr>
</tbody>
</table>

Table 2. Characterization vectors.

### 3.2 Results

Within the WISE project, two process models with 11 and 13 sub-processes, respectively, were compared.

In a first step, similarity facts between products were set by the process engineer according to content and purpose of the documents accessed by the different processes. Since neither tools nor roles were modeled at this stage of the project, heuristics referring to these entities were ignored. In a second step, a computation was performed in order to analyze commonalities among the single processes comprised by the different phases. Finally, another computation, with other weights, was performed in order to analyze commonalities among the different phases of the two development processes. Besides, even the way SPEARSIM heuristics are interrelated forces a concept-based analysis.

The computation, aimed at the analysis of commonalities among single constituent processes, was performed with $P_{CM} = P_{dS} \cdot 1.0$. 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.

The analyses of the commonalities among phases was performed with $P_{CM} = P_{dS} \cdot 0.609 + P_{cH} \cdot 0.138 + P_{cS} \cdot 0.253$. 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).

Figure 6. Similarities between phases.

Figure 6, shows the similarities between phases as computed by SPEARSIM. Since the phases of both models are in a chronological order in the diagram, the disposition of the greatest similarities along the main diagonal (highlighted by the ellipse) confirms the expectation of a certain correspondence between the two models. The diagram shows the greatest commonalities in the requirements as well as in the test phases of the two development processes. A mismatch of the phases development phase (pilot 1) and coding phase (pilot 2) shows where to expect the greatest differences between the two development processes. The main reasons for the differences can be found in different maturity 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 1, and a distributed game implemented in Java in the case of pilot 2. The similarities between the pilot 2 requirements phase and the pilot 1 development phase as well as between the pilot 2 design phase and the pilot 1 requirements phase show commonalities in the interfaces between these phases, i.e., the requirements documents.
Figure 7 shows the similarities computed between the underlying processes, which are arranged on the axes in a chronological order. Although a more complex situation is given here, in this case, most of the greatest similarity values are also arranged along the main diagonal of the diagram. Therefore, most of the relationships shown were expected. The reasons for some exceptions rest upon different granularity degrees of the models as well as on 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 8 shows an excerpt of the reference process model.

This excerpt taken from the WISE project shows merged common processes/products and optional processes. The process engineer merged the commonalities 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 owners and merged into the comprehensive process model as an optional part of the process.
In the WISE project, none of the pilots had an activity for designing heterogeneous clients. As a consequence, a process feasibility study was proposed as part of the software process reference model. Its purpose is to perform a study in order to solve technological questions that arise from the requirements with respect to heterogeneous clients.

Guidelines were incorporated informally as part of the description of each process in the software process reference model. The process engineer de-
scribed each process as a list of steps and a group of decisions that might influence the process.

Each decision has a question and a set of options that are the possible answers. The process owners, when faced with these influencing decisions, were not able to prioritize them. Recentness of the domain was a factor that blurred the view. Refinement of these guidelines is expected as more knowledge from the domain is gained from the pilots. Table 3 gives an example for unit testing guidelines.

| Description: | Test units of code (modules, functions, methods) until defined test coverage is reached; fix all defects found; record defects information. |
| Question: | How can I test the basic functionality of a wireless Internet service client module? |
| Option 1: | Use mobile device emulator environments for testing the functionality of client modules. That is the case with Sun’s “J2ME™ Wireless Toolkit”, which is a set of tools that provides developers with an emulator environment for implementing applications targeted at CLDC/MIDP compliant mobile phones and entry level PDAs. CLDC/MIDP is the Java runtime environment for today’s mobile information devices such as phones and entry level PDAs. |
| Option 2: | Produce code from scratch that complies with specifications of the mobile device where the module will be deployed. |

**Table 3.** Example of informal guidelines.

### 3.3 Lessons learned

In the following, we sketch some lessons learned during the analysis of commonalities and differences of some real software processes:

The case study showed that performing a commonality analysis and doing so in a systematic way, is needed in order to create a meaningful, consistent, software process reference model. This assumption becomes more relevant the larger the compared process models are.

It was also seen that the technique followed for performing the commonality analysis (see Section 2) is a good basis for defining a systematic analysis of commonalities. The method should be further refined.

Another important finding was the usefulness of the SPEARSIM for the commonality analysis. The tool does not only provides great help to the process engineer in handling complexity, but also provides assumptions based on rules.

Comparing entities manually depends on the complexity and size of the process model, and on the knowledge of the process engineer. For complex process models, tool-supported comparisons may support manual analysis. Neverthe-
less, it is still unclear whether all available heuristics are indispensable for comparing process models. Also, as mentioned in the description of SPEARSIM heuristics, in general, the process engineer sets the weights for performing a similarity analysis based upon his experience and process knowledge. More experience with the tool is still needed in order to understand whether different sets of preset weight values can improve the usability of SPEARSIM for specific analysis types, e.g., analysis of commonalities among phases, constituent processes, or products.

The tool demands not only good knowledge of the process models but also a good knowledge of the rules, the weights and of how to combine them. On the other hand was shown that its interface and its results are simple to understand, that it provides assumptions, proposes similarities and differences, and that the process engineer can save the results. Thus, its use can lead to such benefits, as better understanding of the complexity of the model, swift analysis of commonalities and differences, and, therefore, faster creation of the software process reference model.
4 Related work

One approach for capturing a large software development process by integrating views modeled independently is presented by [10]. This approach is based on ideas from other fields like database schema integration and requirements specification. The approach is called Multi-View-Modeling (MVM). Here, the process engineer models the processes by interviewing specialists in concrete situations, or he models the process in a role-oriented way. For each role, a view is developed. Commonalities are analyzed in order to uncover inconsistencies between views of the same process model. Inconsistencies should be fixed in order to have an integrated, consistent, comprehensive process model. This is the main difference to the technique presented in this article. The analysis presented in this article compares different process models of the same domain. Therefore, the term inconsistency cannot be used for the results, but ‘options’ or ‘alternatives’. Although SPEARSIM was developed based on [10]’s approach, it proved to be useful because the rules for comparing processes can also be applied.

A similar method was proposed by [16] for eliciting processes using a view-based approach. A prototype called v-elicit was implemented in order to elicit views in parallel from multiple sources. The steps are basically the same steps as those of the MVM, but it differs on the rules used to detect inconsistencies. A commonality analysis is also performed, however there is no description of how to do it, just the suggestion to compare only the key activities.

A similar problem, integrating schemas of existing databases from the perspectives of different users (database schema integration) is addressed by [2]. The formalism used to express database schemas separately and integrated is the same. This makes integration easier than in the MVM approach or in the technique presented in this article. 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. Every 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 [6], [7], [9]. These approaches have in common that they integrate pairs of specifications, 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 done, 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 approach, there has also been research dealing with commonality analysis [12]. At a high abstraction level, the steps to be followed for creating a family of products are basically the same as the ones proposed here for creating the reference models. On the other hand, comparing the products of two process models in order to see if their purpose is the same is a different activity as comparing them in order to see if functions, procedures or components can be reused. 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. This step can be compared with the definition of the context vector before comparing process models.

Identify commonalities and variabilities: 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 variabilities: A range of values for the variants is defined.

Exploit commonalities and accommodate variabilities: The results from the commonality analysis are grouped into procedures, inheritance, and parametric polymorphism, which, is similar to how the models are integrated in the technique presented here as common, alternative and optional parts.
5 Summary and outlook

The article presented a tool-supported technique to handle process diversity in the software development domain. Effective and predictable software development requires the existence of customized processes that are based on empirical experience in the field. The existence of a process understanding should be a prerequisite for the creation of reference process models.

The application of the technique in the wireless Internet service domain showed that a systematic analysis of commonalities and differences between process instances is an indispensable step towards the creation of a software process reference model for a specific domain. An advantage of the presented bottom-up approach (i.e., the derivation of the reference process model from pilot processes) is the closeness of the resulting reference model to practices in real life. Nevertheless, creating a reference model only from observation of software projects limits the level of detail and precision to the abilities of the respective project organizations in which the projects are performed. Therefore, it is recommended to enhance the developed reference model carefully with knowledge from other sources (e.g., literature, experience reports from similar projects).

The evaluation of the technique revealed that appropriate notations for describing generic process knowledge (i.e., adaptable process models and adaptation rules) are widely missing. Only very specialized notations are available; appropriate graphical notations and tool support for representing them are missing. Optional product flows, for instance, are difficult to represent in existing notations.

Another experience is that, especially in new domains, there is a lack of knowledge about the impact of context characteristics on the process design and its behavior. Empirical software engineering methods (such as performing controlled laboratory experiments, or the use of the existing body of knowledge from empirical studies) might help to better understand this relation. We suggest that a combination of qualitative and quantitative analysis will help to describe this impact more clearly. Finally, the evaluation has shown that SPEARSIM can support the commonality analysis.

Future research may address the following questions: What are appropriate notations for representing generic processes? How to determine effectively the impact of context factors on processes? Which degree of process complexity requires automated similarity analysis? What are appropriate rules to determine process similarity and when to apply them?

Future work in the WISE project comprises the iterated application with further case studies.
Acknowledgments

The 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 for the fruitful cooperation, especially the team members from Motorola Global Software Group - Italy (Motorola GSG-Italy), and Investnet (Italy). We would like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first version of the article. Additionally, we would like to thank the anonymous reviewers for their helpful comments on this article.
References


