Software Process Commonality Analysis

Alexis Ocampo, Jürgen Münch, and Fabio Bella

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 of these 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.

Index Terms— Software engineering process, software process models, variability, commonality.

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 its 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 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 (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

• A. Ocampo, J. Münch, and F. Bella are with the Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, 67661 Kaiserslautern, Germany. Email: {ocampo, muench, bella}@ ise.fraunhofer.de.

Manuscript received 25.02.2004.

xxxx-xxxx/0x/$xx.00 © 200x IEEE
systematic way, in order to understand their context-dependent variabilities.

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

In the product line world, identifying commonalities and differences is an accepted, widespread practice when comparing systems [4]. Usually, common elements are reused and variabilities are hidden, in the most appropriate way, in order to produce a family of products.

In the process modeling world, there exist some approaches to integrating partial process models (i.e., views) into a descriptive process model when multiple persons describe their perspectives of a large software process [5], [6]. These approaches are called multi-view modeling approaches. 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.

Also 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 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).

The technique was developed and evaluated as part of the WISE project. 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 empirical approach described here, with the observation of techniques and processes followed during realistic pilot projects being a key activity.

3 TECHNIQUE

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

Manual Commonality Analysis

Manual commonality analysis applies to a set of representative process models for the same domain. The models must be rigorous, and this may be achieved, for example, by using electronic process guide (EPG) capabilities having graphical views [7]. The technique for manual commonality analysis provides guidelines on how to compare the process models and identify structurally similar or different process parts. 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, Fig 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.

**Tool-supported Commonality Analysis**

The tool SPEARSIM is designed to support a process engineer in comparing large and complex processes. This tool analyzes the similarity of two process models using specific 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. Fig 2 shows an excerpt of the table of commonalities after the process engineer has turned all the assumptions into facts. This table of commonalities could be used in reviews with developers to obtain a common agreement. 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 Fig 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.

Fig 3 presents a view, generated by SPEARSIM, showing the similarities between phases. Because 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 expected correspondence between the two models, which was observed in the preliminary manual analysis. 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 Fig 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 game implemented in Java in the case of pilot project 2. Although this is an obvious result, 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 then can nominate as ‘best practices’ that should appear in any customized version of the reference process. Therefore, going back to the example, our proposal will then be that a new project intended to develop a software product in the wireless Internet services domain should have as part of its process a subprocess with the scope and characteristics of the acceptance test.

**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 further research how to specify additional kinds of rules, in particular rules followed intuitively by process engineers, in real practice, to determine best practices and acceptable/unacceptable process variations. 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 positive and negative process variations, might be valuable for software organizations in other areas, 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.

ACKNOWLEDGMENT
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 Narmingha 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., Mönch, 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., Mönch, 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] Coplien, J., Hoffman, D., Weiss, D.: Commonality and Variability in Software Engineering. IEEE Software, vol. 15, No. 16, pp. 37-45, Dec 1999.
[5] Turgeon, J., Madhavji, H., N.: A Systematic, View-Based Approach to Eliciting Process Models. In proceedings European Workshop on Software Process Technology, pp. 276-282, 1996.
[6] Verlage, M.: An Approach for Capturing Large Software Development Processes by Integration of Views Modeled Indepently. Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering SEKE, 1998.
[7] Becker-Kornstaedt, U., Scott, L., Zettel, J.: Process Engineering with Spearmint/EPG. Proceedings of the 22nd International Conference on Software Engineering ICSE, 2000.