Towards a Process-Line for MDPLE

Maider Azanza[1], Josune de Sosa[2], Salvador Trujillo[2], Oscar Díaz[1]

[1] University of the Basque Country. San Sebastian, Spain
{maider.azanza, oscar.diaz}@ehu.es
[2] IKERLAN Research Centre. Mondragon, Spain
{jdesosa, strujillo}@ikerlan.es

Abstract. The conception and design of software-intensive systems is an inherently complex endeavor. We experienced this complexity ourselves while engineering a wind turbine control system. Such complexity was faced by the combined use of Software Product Line Engineering and Model-Driven Engineering. However, the application of both paradigms demanded considerable changes in the software development process. This position paper focuses on the process we followed in engineering the system and reports some experience. Overall, this experience advocates for the explicit definition of Model-Driven Product-Line Engineering based Processes. In certain domains, such processes need to be fine-tuned to accommodate specific customer or project needs and thus the notion of process-line for MDPLE is introduced.

1 Introduction

The software industry remains largely reliant on the craftsmanship of skilled individuals engaged in labor intensive manual tasks. However, growing pressure to reduce cost and time to market, and to improve software quality, may catalyze a transition to more automated methods [9]. *Model Driven Engineering (MDE)* and *Software Product Line Engineering (SPLE)* are two paradigms aimed at industrializing the software development process.

The main focus of MDE are models, which will then be transformed to concrete implementations [8]. MDE raises the level of abstraction, thus permiting developers to concentrate on the essential domain concepts, decoupling them from concrete implementation details. As for SPLE, it aims at building a set of related products out of a common set of core assets.

The benefits of MDE and SPLE have been widely reported, including different industrial case studies [3,5]. Being complementary in nature, the combination of both, which is referred to as *Model Driven Product Line Engineering (MDPLE)*, integrates the advantages of both [18]. Hence, MDPLE has been subject of research in the recent years [1,14].

Nevertheless, the adoption of MDE and SPLE, and furthermore MDPLE, requires considerable changes in the manner software development is carried out, which is translated into an up-front investment that needs to be considered. An assesment of the implications of the adoption is essential [17].

In this sense, the definition of a rigorous process that specifies how to realize these paradigms is necessary to fully exploit their advantages [7]. As a case in point, in a previous work we reported the challenges MDPLE brings for assembly processes [2].

In this paper we report our experience when developing wind turbine control systems in the wind power industry [19,20]. Such a system is subject to a growing market and technology challenges, which motivated its reengineering following MDPLE principles. We describe the general process we followed and motivate the need for a explicit and systematic process if MDPLE is to achieve its full potential. Nonetheless, different projects or domains may have different requirements. Thus, the need for a variable process that can be fine-tuned to each of them is explained and the variability sources we identified in our case study are described. A process line is introduced as possible solution to manage such variability.

2 Motivating Case Study

2.1 General motivation

Software has evolved from relatively small and simple products to systems with a considerable size and a high degree of complexity. Extensibility and customization to each customer are nowadays a requirement more than a wish. At the same time, the product's desired time-to-market is ever decreasing. To tackle these needs, advanced software paradigms have emerged. *Model Driven Engineering (MDE)* and *Software Product Line Engineering (SPLE)* are two cases in point.

MDE raises the level of abstraction, defining models that capture the specifics of the application at hand, which will then be transformed into the actual application code implementation. MDE leads to conceptual simplicity: clear architecture, efficient implementation, higher scalability and greater flexibility [4].

As for SPLE, it aims at building a set of related products out of a common set of core assets. Unlike MDE, now the stress is not so much on the abstraction level at which software is specified, but on conceiving programs as pre-planned variations from core assets. The definition of a set of related products in the form of a product family increases reuse and reduces the cost of each of them.

Both MDE and SPLE depart from one-off development to provide an infrastructure where different (though related) products can be obtained. The benefits of applying them separately in an industrial setting have been reported in the literature [3,5]. Their complementary nature permits their combination in MDPLE that brings even greater benefits. We have experienced this ourselves [19,20].

However, MDPLE requires considerable changes in an organization and in the processes that yield each product. Compared to traditional development processes, new activities, tasks and artifacts emerge, which need to be explicitly defined beforehand if all the expected benefits are to be obtained.

The following section briefly describes the specific challenges related to the process when engineering wind turbine control systems.

2.2 Case Study: Green Energy Control Systems

In many domains, the general motivation outlined above occurs. Indeed, this is the case when developing wind turbine control systems in the wind power industry [19,20]. A multidisciplinary team of engineers work together in the development of the embedded control system [12]. Such system is subject to a growing market and technology challenges. Some of them impact directly on the process.

- *Stringent Time-to-market.* The challenge is twofold: new wind turbine models and customizations of wind turbine models for specific systems are needed. In the first place, the required time between releases to market models is reducing. Hence, there is a need to accelerate the process to conceive and design newer system models. In the second place, there are different customer expectations and needs for each system, that have to be fulfilled. Both trends introduce pressures to the acceleration of the development process.
- *Industrialize the process.* Raise efficiency, automate and industrialize the development of systems. The growing sector of alternative energies is facing a continuous growth together with an internationalization of operations worldwide. Hence, there is not only a unique product to be managed, but a whole family in different geographical locations. The challenge is to industrialize and geographically distribute the process to create and customize control systems worldwide.
- *Scalability and Teamwork.* The size and number of elements to be controlled is growing. The knowledge is distributed among different engineers from different disciplines. Hence, the need to organize workteams, define roles and responsibilities, and explicitly define a process.

In general, all these challenges call for the need to explicitly define a process for model-driven product-lines. Besides, there is a growing need to customize such process to the needs of different systems or even application domains.

First, we describe the process we followed. Then, we argue on the variability of such process.

3 An MDPLE-Based Process

A general overview of the process to create a product following MDPLE is given in this section. The process that realizes each activity has to be detailed next. As an example, the process that builds the model driven infrastructure is presented.

3.1 Overview

We describe the general MDPLE process next. First, the general development process was divided into coarse grained activities. Note that the development

process required several iterations, if new features had to be incorporated or errors were discovered the activity had to be executed again. A brief description of the activities we identified follows:

- *The Art of Decisión Making (DEC).* Given the requirements of a certain project, a financial assessment is made and the necessary resources are allocated. It is a managerial activity, which has a direct impact on but does not per-se belong to the software development process, but an evaluation of whether the up-front investment an MDPL requires is worthwhile is essential.
- *Family Management for the Infrastructure (M2PL).* The benefits of SPLE can be extended beyond the product family. The development of metamodels and transformations is a cumbersome task which can benefit of increasing reuse. This activity defines a family of metamodels and transformations that permit to increase reuse by applying SPLE techniques [22,25]. The resulting products of this activity will be the MDE infrastructures (i.e. metamodels and transformations), that will later permit to obtain the actual products. This activity corresponds to domain engineering applied to MDE infrastructures. Note that the effort needed to develop a family may not pay off in all cases (e.g. when metamodels and transformations have no variability). This is an indication that some of the activities in the process are optional.
- *Build the Infrastructure (M2Dev).* The aim of this activity is to obtain the concrete metamodels and transformations (i.e. the model-driven infrastructure) that will be later used to create individual products. In this case, it corresponds to application engineering for the MDE infrastructure. We detail this activity in the following section.
- *Adapting the Infrastructure to the Specific System Domain (M2Adapt).* Requirements that are not accounted for in the shared metamodel and transformations may be accommodated by this adaptation activity that tailors them to the specific domain's needs [6].
- *Family Management for the System (M1PL).* This activity refers to the traditional domain engineering. It involves identifying commonalities and variability between product family members, and implementing a set of core assets liable to be reused along different products.
- *Build the individual System (M1Dev).* This activity performs application engineering. It yields the final system using both the metamodel and transformations developed in *M2Dev* and the core assets that were produced in *M1PL*.

3.2 M2Dev: Build the Infrastructure

The purpose of this section is to detail the subprocess *Build the Infrastructure (M2Dev)* for our case study. The roles include: the *chief engineer* who takes the decision to use the MDE paradigm, the *chief architect* who architects the overall solution, the *modeling engineer* who analyzes the structure and abstractions of the original code, the *programmer* for the alignment of the original system, the *metaware engineer* who creates the metamodels and transformations, and the

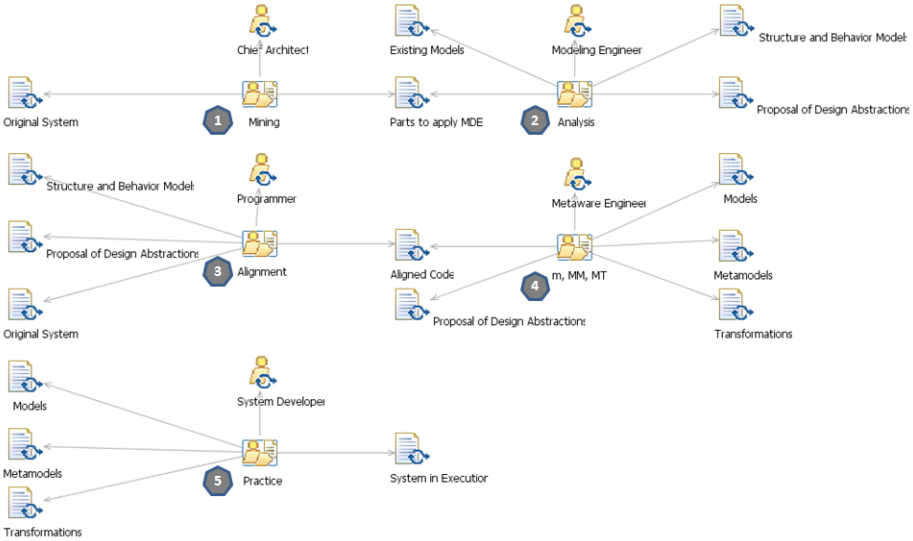

Fig. 1. *M2Dev* process detail in SPEM

system developer in charge of the development of individual systems. Figure 1 sketchs an overview of the process using SPEM [13]. Next subsections outline the activities.

Mining Once the paradigm was selected, the next step was a preliminary analysis of the existing system. This is an important point. While there exist processes that generate the system's software from scratch [11,15], in our experience it is more usual to depart from an existing system that is already in operation. The goal of this activity was to decide to which parts MDE can be applied. The first step was to identify parts of homogeneous or potentially repetitive code (code clones). Note that already existing behavioural and structural models largely supported our analysis. However, code was not automatically generated from such models. The search and selection of candidate parts for automation was based on the abstractions we manually found[3]. The *chief architect* led a multidisciplinary team to find those candidates. The mining resulted in a focus on a subset of subsystems, whose implementation code was to be automated from models. These parts largely changed among products due to customer requirements. Automating the code generation will ultimately reduce the development and maintenance efforts.

[3] This was not automated since specific domain knowledge was required. Hence, further work may investigate the automation of the extraction of these abstractions by following an Architecture-Driven Modernization (ADM) approach.

Analysis A detailed analysis on selected parts was conducted to examine in detail the code to be generated. The previously existing models played a pivotal role. We actually enriched such models with further information needed to really generate executable code. For instance, it is not the same the definition of a condition from a design perspective than the precise definition needed for such condition to be executable. The *modeling engineer* was responsible for conducting the analysis of the structure, the behavior and the abstraction. The level of detail in the models was largely increased from simple sketches to rather complete models. We also completed the structure and behavior models for all code parts.

Alignment The analysis provided a detailed knowledge on the models and code. A last step was needed in our case before automating code generation. It involved the alignment of the implementation code. For alignment, we understood for the code to be implemented in the same way in similar situations. For example, we detected that some statecharts were implemented following a *switch-case* style, whereas others followed an *if-else*. This and other details were homogeneized. In general, code was prepared to be always *generated* in the same way. This impacted following the same structure, the same methods name, same convention to define attributes, etc. This alignment of code and the unification of the programming style were critical to automate code generation. The *programmer* was responsible of leading this task. A requisite in our project was to ensure that engineers understood *the generated code* as they understood the previous manually implemented code. Eventually, the alignment allowed the code to be automatically generated from a model.

Models, Metamodels and Transformations (m,MM,MT) UML was also used as a metamodel. Our models were mostly sketched as class and statechart diagrams. In some cases, we also introduced some stereotypes to ease automatic code generation [20]. The level of expressiveness drove such different modeling decisions [23].

Practice The completion of the infrastructure preceded its deployment for models, metamodels and transformations to be used in practice for the development of individual systems. To attain this, the internal leadership is essential in the customer organization. The *system developer* was in charge of this. Besides, a detailed process was set in place. In our case, this leadership was possible by the implication of the customer engineers in some tasks of the previous activities. This enabled him to familiarize with the new way of working and the new tools.

4 Towards an MDPLE Process Line

This section analyzes whether the same process can be used for different systems within the organization or even beyond the organizational boundaries. The

notion of process-line for MDPLE process as a product-line of process is introduced.

4.1 Motivation

An MDPLE process has to be general enough to allow reuse in different domains, systems and projects. The first approach is to define a comprehensive process embracing the entire set of activities that can occur in every situation, and then, provide the choices within the process. However, this enlarged approach has a number of limitations, namely, the delay in the process introduced by taking again decisions that were already decided, and by carrying out activities that can be unnecessary.

An alternative approach could be the definition of a generic process that is customized for each project. Doing so, the process is customized before starting it. In this customized process, only necessary activities are carried out. Hence, time is significantly reduced by removing unnecessary activities and decisions [10]. As a next step, the definition of all related processes in an organization as a family can bring the benefits of SPLE to the process realm [16]. However, note that the definition of a process line (i.e. the core assets that yield the product family) is not a trivial task. Such process line would permit us to systematically yield process instances for each particular domain (e.g. some may require only MDE, others SPLE and others the combination of both).

Next, we introduce the different kinds of variability that can appear in our projects.

4.2 Variability of the Process-Line

The process often needs to be customized to meet the needs of different customers. A *one-size-fits-all* is inappropriate because of the large variability involved. Next we present the variability that originate in the process to subsequently sketch how to address them.

- *Different project size.* Depending on the size of the system, the project size varies largely ranging from individual (1 person), small team (below 10), to larger projects (above 10). In small projects the application of a full-fledged MDPLE process may not be desirable, or even affordable.
- *Different levels of automation.* There are different levels of automation in code generation. There are some MDE projects where models are mainly used for design, whereas others models automate code generation. Besides, there are different sorts of transformations, namely, model to model and model to text transformations. Overall, there is a variety of approaches.
- *Different tools.* Market diversity provides a variety of modeling tool support with different characteristics that come at different cost, which can be imposed by different customers.
- *Different artifacts that cope with variability.* Variability is incorporated in the models. Increasingly, variability is also spreading to metamodels and

model transformations [21,25]. This relates to the coarse grained activities described in Section 3.1.
- *Different project infrastructure.* The resources involved in different projects vary among teams. Assorted technologies for repositories, use of wikis or other tools may differ.

These differences among projects involved that the process needs some sort of customization. In particular, a number of activities and some decisions are bound to such differences. The specific mechanisms used to handle variability are out of the scope of this paper [19]. Overall, they are those commonly found in software product-lines. Feature models for representing the variability in the whole family, variation points in the specification, and mappings connecting features to variation points.

The sources of variability for the process-line vindicates the need to accommodate the MDPLE process in order to achieve economies of scale not only on the system, but also on the process.

5 Related work

Previous work describes the benefits of applying MDE and SPLE in industrial case studies, including embedded systems [3,5,15]. Among the benefits they report flexibility, better adaptation to change and an increase in productivity and product quality can be highlighted.

Nevertheless, the adoption of MDE and SPLE in isolation, not to mention MDPLE, in a project is far from trivial. It involves an up-front investment and considerable changes in the organization. As an example, a list of requirements for MDE to be successful are described in [17]. In our limited experience, the need to ponder the benefits of applying each paradigm is vital and we have reflected it in the first activity of our process (i.e. DEC).

As mentioned before, the contributions of MDE and SPLE have been already documented. Their combination brings even greater gains [18,24]. However, the process that leads to obtain each product, and thus that brings such gains, incorporates new activities, roles and artifacs, increasing its complexity when compared to traditional software development. The need to define a process that systematizes the development of the model driven infrastructure was presented in [7], where the requirements such process should fullfil where described. Based on an industrial case study, we have defined a preliminary process that targets those needs (see Subsection 3.2). As a case in point, in a previous work we reported the challenges MDPLE brings for assembly processes [2].

Examples of processes for embedded systems can be found in [11,15]. In both cases, the infrastructure to produce a product is already given. Our process includes the development of such infrastructure, as it can vary from domain to domain. Moreover, both processes create systems from scratch. In our experience, it is more common to have an existing system that has to be reenginered to apply MDPLE.

Process customization, process variability and process families are an emerging topic [10,16]. We described the variability we encountered when developing an MDPLE-based system and sketched the sources of variability we encountered.

6 Conclusions and Future Work

This paper reported our practical experience applying Model Driven and Software Product Line engineering to a family of wind turbine control systems. Overall, we introduced the MDPLE-based engineering process we followed. The activities for building the MDE infrastructure were detailed.

We also focused on the need for variability for the process itselft and the sources of variability we identified for such process-line were also described. The definition of a process line that permits to manage such variability by yielding a family of MDPLE processes is subject of future work.

Our work paves the way for further research on the field. Indeed, we are currently working on detailing the entire process and on the specific mechanisms to handle the process-line variability.

Acknowledgments. This work was co-supported by the Spanish Ministry of Education, and the European Social Fund under contract MODELINE, TIN2008-06507-C02-01 and TIN2008-06507-C02-02.

References

1. 1st International Workshop on Model-Driven Product Line Engineering (MDPLE), 2010. http://www.feasiple.de/workshop_en.html.
2. Maider Azanza, Oscar Díaz, and Salvador Trujillo. Software Factories: Describing the Assembly Process. In *International Conference on Software Process (ICSP)*, 2010.
3. Paul Baker, Shiou Loh, and Frank Weil. Model-Driven Engineering in a Large Industrial Context - Motorola Case Study. In *International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 476–491, 2005.
4. Jean Bézivin. On the Unification Power of Models. *Software and System Modeling*, 4(2):171–188, 2005.
5. Paul Clements and Linda M. Northrop. *Software Product Lines - Practices and Patterns*. Addison-Wesley, 2001.
6. Sybren Deelstra, Marco Sinnema, and Jan Bosch. Experiences in Software Product Families: Problems and Issues During Product Derivation. In *International Software Product Line Conference (SPLC)*, pages 165–182, 2004.
7. Frédéric Fondement and Raul Silaghi. Defining Model Driven Engineering Processes. In *International Workshop in Software Model Engineering (WiSME)*, 2004.
8. Robert B. France and Bernhard Rumpe. Model-Driven Development of Complex Software: A Research Roadmap. In *Workshop on the Future of Software Engineering (FOSE 2007)*, 2007.
9. Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.

10. Peter Killisperger, Markus Stumptner, Georg Peters, Georg Grossmann, and Thomas Stückl. Meta Model Based Architecture for Software Process Instantiation. In *International Conference on Software Process (ICSP)*, 2009.
11. Ali Koudri, Joël Champeau, Denis Aulagnier, and Philippe Soulard. Mopcom/marte process applied to a cognitive radio system design and analysis. In *European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA)*, pages 277–288, 2009.
12. Jonah Z. Lavi and Joseph Kudish. *Systems Modeling and Requirements Specification Using ECSAM: An Analysis Method for Embedded and Computer-Based Systems (ECBS'04)*. IEEE Computer Society, 2004.
13. OMG. Software Process Engineering Metamodel Specification. Formal Specification, April 2008. Online at: http://www.omg.org/spec/SPEM/2.0/PDF.
14. AMPLE Project. Aspect-Oriented, Model-Driven Product Line Engineering, 2010. Online at: http://ample.holos.pt/.
15. Wilhelm Schäfer. A Rigorous Software Process for the Development of Embedded Systems. In *International Software Process Workshop (SPW). Revised Selected Papers*, pages 91–99, 2005.
16. Borislava I. Simidchieva, Lori A. Clarke, and Leon J. Osterweil. Representing Process Variation with a Process Family. In *International Conference on Software Process (ICSP)*, 2007.
17. Miroslaw Staron. Adopting Model Driven Software Development in Industry - A Case Study at Two Companies. In *Internationl Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 57–72, 2006.
18. Salvador Trujillo, Don Batory, and Oscar Díaz. Feature Oriented Model Driven Development: A Case Study for Portlets. In *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May*, 2007.
19. Salvador Trujillo, Jose M. Garate, Roberto E. Lopez-Herrejon, Xabier Mendialdua, Albert Rosado, Alexander Egyed, Charles W. Krueger, and Josune De Sosa. Coping with Variability in Model-Based Systems Engineering: An Experience in Green Energy. In *European Conference on Modeling Foundations and Applications (ECMFA)*, 2010.
20. Salvador Trujillo, Jose M. Garate, Xabier Mendialdua, Albert Rosado, and Josune De Sosa. The Future of Green Energy Control Systems. In *Draft under Review*, 2010.
21. Salvador Trujillo, Ander Zubizarreta, Josune de Sosa, and Xabier Mendialdua. Is Model Variability Enough? In *1st International Workshop on Model-Driven Product Line Engineering (MDPLE)*, 2009.
22. Salvador Trujillo, Ander Zubizarreta, Xabier Mendialdua, and Josune de Sosa. Feature-Oriented Refinement of Models, Metamodels and Model Transformations. In *1st International Workshop on Feature-Oriented Software Development (FOSD)*, pages 87–94, 2009.
23. Markus Völter. MD* Best Practices. *Journal of Object Technology (JOT)*, 8(6):79–102, 2009.
24. Markus Völter and Iris Groher. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In *International Software Product Line Conference (SPLC)*, 2007.
25. Jules White, James H. Hill, Jeff Gray, Sumant Tambe, Aniruddha S. Gokhale, and Douglas C. Schmidt. Improving Domain-Specific Language Reuse with Software Product Line Techniques. *IEEE Software*, 26(4):47–53, 2009.