Experiences in Teaching Variability Modeling and Model-driven Generative Techniques

Philippe Collet, Sébastien Mosser, Simon Urli, Mireille Blay-Fornarino, Philippe Lahire
Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France
{collet, mosser, urli, blay, lahire}@i3s.unice.fr

ABSTRACT

Teaching software engineering is an activity that needs to constantly evolve to cope with new paradigms, principles and techniques. In this paper, we report on several years of experience in teaching both generative techniques in a model-driven engineering context and variability modeling related to software-product line engineering. Our current practice relies on making students progress on running projects that they evolve with different techniques along a semester. We also discuss the obtained benefits and some perspectives.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education; D.2.13 [**Software Engineering**]: Reusable Software

1. INTRODUCTION

Teaching software engineering is an activity that needs to constantly evolve to cope with new paradigms, principles and techniques [4]. Software Product Lines (SPL) is a software engineering paradigm which is under development for more than twenty years now. It has notably been successfully applied in software-intensive systems to support the development of a family of similar software products from a common set of shared assets [3]. SPLs are now commonly adopted in industries in order to optimize the reuse of quality software components and many different software application domains are also switching to SPL techniques.

As pointed by Acher et al. [2], teaching SPL techniques has not been discussed nor reported enough, while educating the next generation of software engineers to innovative techniques is a crucial need. It is indeed not clear how SPL and related techniques are taught and their survey shows a diversity of approaches and different open issues.

In this paper, we report on several years of experience in teaching both generative techniques in a model-driven engineering context and variability modeling related to software-product line engineering. The courses are positioned in a two-year postgraduate programme at the University of Nice - Sophia Antipolis in France (cf. Section 2). These courses are highly focused on the technical aspects of engineering, and even if domain engineering is introduced within the SPL context, management aspects are not covered. We describe the content of courses and their evolution over several years, leading to this year organization with a focus on making the students progress on running projects that they evolve with different techniques along a semester (cf. Section 3). We also discuss feedback, obtained benefits from these years of teaching, and some perspectives.

2. CONTEXT

The reported experience concerns several software engineering courses that range over the two years of the postgraduate programme in the University of Nice Sophia Antipolis. Over 6 years, the software engineering course at the first year of MSc (M1) has evolved from a full semester course to a split version with two third of software engineering and an optional course (on SPL techniques). This year, due to new organization rules at the MSc level, it was again a full semester course. At the second year (M2), the teaching has been highly focused on model-driven engineering and generative techniques for seven years. For several years, two full semester courses[1] were dedicated to Model-Driven Engineering (MDE) concepts and advanced applications (embedded systems, etc.).

3. TEACHING EVOLUTION OVER YEARS

We present here the content of the different courses following a chronological order and focusing on the parts related to SPL, variability or generative techniques.

3.1 First MDE Courses

The first introduction of a SPL related part in the software engineering courses was done in 2010 at the second year level (i.e., M2, upper part of figure 1). It was organized as a 4 hours seminar on software factories, showing the overall process, the usage of generative toolchains and relationships with model-driven techniques that were also presented in the course. The MDE teaching has itself started in 2008, but was split into two courses in 2010, with a first part focusing on metamodeling and model-to-model transformations.

[1]Actually, courses at the M2 level are slightly shorter, as semesters are themselves shorter to accommodate a final internship.

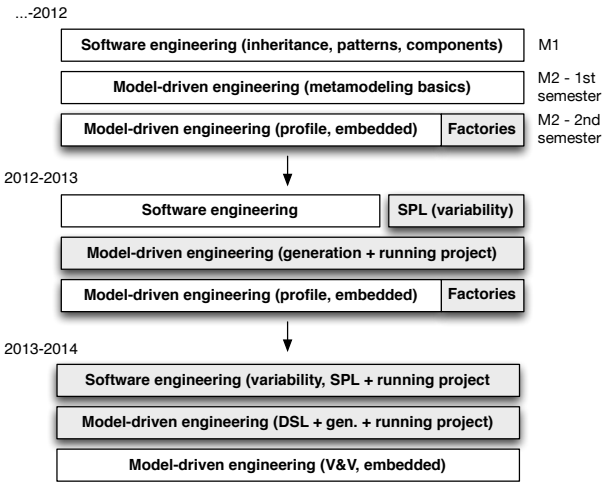

Figure 1: Teaching Evolution

The second part was strongly focusing on profile mechanisms with an application to embedded systems, MARTE and SysML were typically evoked in this course.

This first step typically suffers from some issues identified in the survey from Acher et al.[2]. The software factories course was at a very high level as *case studies and tools were not available* or not easily integrated in the rest of the teaching curriculum. It was also clear for us that the *complexity of the subject* was quite high for the audience. Besides, we noted increasing difficulties for students to build the metamodeling abstraction on small examples or medium-size case studies (at the scale of both courses at the M2 level).

3.2 Introducing Variability Modeling

In 2012, the reorganization of the course curriculum at the M1 level enables us to introduce a dedicated optional course on SPL engineering. The aim of these optional courses was to provide some advanced notions to first-year graduate students while not being a prerequisite for any M2 level course.

Still facing the lack of case studies and end-to-end SPL tool, we focused this first course on a general introduction on SPL concepts (domain and application engineering, assets, consistency issues, configuration and derivation processes), while using the FAMILIAR DSL[1] for lab exercises. These exercises dealt with variability modeling from software artifacts, using feature models. A class project was about modeling the variability of a part of the twitter API, which we identified as a good candidate for showing the need for the expressiveness of feature models. The exam itself consisted in creating a feature model reproducing a car configurator. This exam was also a way to conduct a controlled experiment as the students had to build the feature model with FAMILIAR with a new graphical interface and the scripting console.

The course result and feedback from students show that the feature modeling concepts were mainly understood and acquired. Still the course itself lacked a broader application of the SPL concepts and a clearly demonstrated relationship with other software engineering issues and solutions.

In the meantime the first MDE course at the M2 level was switching to an approach relying on an end-to-end project covering the creation of metamodels, OCL constraints and appropriate transformations to manage the visualization of sensor data. While showing progress in the understanding of a complete MDE toolchain by students, the used case study was still quite abstract to demonstrate clear benefits in abstracting away from technological particularities and implementation details.

3.3 Last Project-centric Organization

In 2013, a new reorganization at the M1 level made the software engineering course back at a full semester size including the SPL part, being now taught to all graduate students. This was an opportunity to try to smoothly integrate the SPL paradigm in this software engineering course. We relied on an existing running project which case study consists in evolving and improving a simulator of *creatures*, based on fixed-time steps engine and developed in Java. This case study already enabled us to progressively introduce different software engineering concepts:

- Code quality and refactoring (starting with a badly designed system);
- Unit testing, automation and coverage (improving an existing test suite);
- Inheritance and composition issues (abstracting simulation concepts, decoupling GUI, etc.);
- Design patterns (organizing different creatures' strategies and decorations, using factories for family consistencies, etc.);
- Programming by interfaces and plugins (refactoring the simulator architecture, making elements dynamically loadable as plugins).

One advantage is that students are more or less taking the previous lab output as the starting point of the next lab. Providing total or partial lab solutions enables to provide well-defined stages to help students in difficulty.

We thus integrated the SPL part as a follow-up of this project. The overall introduction to the SPL paradigm was quite similar to previous year, but parallels with the project case study were systematically described during lectures. At the lab level, variability modeling was again done with the FAMILIAR DSL, but the feature model was expected to cover the variability of the creature simulator, finally organized as a SPL. The connection from the variability model to the software assets was implemented by a bridge using the FAMILIAR Java API. A working code skeleton was provided to students so that they can focus on the mapping and realization implementations. The best groups of students succeeded in building a feature model that separates the domain variability (high-level features from the simulator) from the technical variability (plugin mechanisms, etc.) and also in componentizing some software parts with the Maven automation tool to experiment with different realization mechanisms. Feedback from the students shows a good understanding of both the complexity of large-scale software systems and advantages of switching to the SPL paradigm.

As for the MDE course, this year was also the occasion to refocus the first MDE course on Domain-Specific Language (DSL) engineering so that the metamodeling part is shown as a real support for domain-driven engineering. Moreover the whole lab part was also organized on a running project with a new case study based on the provision of a toolchain to easily program Arduino Uno micro-controllers. Arduino is an open-source hardware platform with an active community[2]. The platform already provides a simplified language, based on the *Processing* programming language, so to program the micro-controllers at a higher level. The concrete aspect of the project was reinforced by providing real Arduino material to all students. They had to assemble micro-controllers on boards with different sensors (e.g., temperature, light) and actuators (e.g., buzzer, led, switch), and to really flash the resulting program, aka. sketch, so to observe results on the assembled Arduino prototypes. Beyond programming, the MDE toolchain also had to support consistency checking on the modeled states and generation of blueprints to help with the Arduino physical assembly.

The project was following a continuous integration approach, obliging students to have as soon as possible a working toolchain from a first metamodel with code generator to reach the Arduino specific language. This first work defines a kernel that as the minimal added-value for leveraging DSLs in the Arduino programming context. Constraints over the metamodel and new functionalities were then incrementally added. As a final project, different groups of students had to extend their MDE toolchain to support more complex Arduino elements and scenarios. The lab sessions used to drive the project are defined in an "a la carte" way, including up to 19 different extensions to be added to the kernel, *e.g.*, supporting new hardware kit descriptors, remote communication, supporting analogical devices, composition of programs, multi-valuable devices (3 dimensional joysticks), macros syntactic expansion, behavioral verification. The concrete realization and the focus on *useful* abstractions were identified as very beneficial to the overall understanding of both the motivations and principles of MDE and generative techniques. In addition, for each implemented extensions, a report was expected to identify what was added into the kernel for this specific extensions.

Finally, with this focus on DSL engineering on the first M2 course, the second course was able to refocus on validation and verification using MDE techniques and targeting embedded systems.

4. CONCLUSION & PERSPECTIVES

In our specific context, we observed it is indeed very difficult to introduce SPL engineering techniques in a software engineering course. More than the known issues about lack of case studies and tools, we identify the problems of showing the complexity of the subject and the needed background on a software system (also reported in [2]) as crucial. We tamed these problems by using running projects in which the complexity is progressively introduced and students get more experienced by evolving the same software system. In both projects, SPL and MDE/generative techniques, we observed clear benefits in motivating students and making them understand both general principles and real applications at a small scale.

Still our current organization can be improved. The separation between the introduction of SPL techniques and the generative techniques based on DSL into two courses is not ideal. It is mainly driven by our specific context of teaching. Nevertheless the project on Arduino micro-controllers is clearly a highly motivating case study that could encompass the majority of illustrations, lab/project exercises we have currently in mind. It could notably been used as SPL case study, with an interesting domain variability in the various possible scenarios and the different kinds of Arduino elements to take into account with their constraints, but also some technical variability with the diversity of Arduino vendors providing almost similar devices.

Another aspect that we would like to cover in the future is the ability to show where DSL, MDE and SPL techniques can be, or should not be, used, separately or together.

Acknowledgments

The work reported in this paper was partly funded by the PING action[3] of the GDR GPL (CNRS, France). Its aim is to put together, at the national level, *recipes* on how to better teach software engineering, with both recipes on concepts and full course organizations.

5. REFERENCES

[1] M. Acher, P. Collet, P. Lahire, and R. B. France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6):657 – 681, 2013. Special section: The Programming Languages track.

[2] M. Acher, R. E. Lopez-Herrejon, and R. Rabiser. A survey on teaching of software product lines. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, VaMoS '14, pages 3:1–3:8, New York, NY, USA, 2013. ACM.

[3] P. Clements and L. M. Northrop. *Software Product Lines : Practices and Patterns*. Addison-Wesley Professional, 2001.

[4] C. Ghezzi and D. Mandrioli. The challenges of software engineering education. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 637–638, New York, NY, USA, 2005. ACM.

[2] `http://www.arduino.cc/`

[3] `http://ping.i3s.unice.fr`