Teaching Software Product Lines: A Snapshot of Current Practices and Challenges

Mathieu Acher, Roberto Erick Lopez-Herrejon, Rick Rabiser

Teaching Software Product Lines: A Snapshot of Current Practices and Challenges[1]

Mathieu Acher, University of Rennes 1, IRISA/Inria, Rennes, France
Roberto E. Lopez-Herrejon, École de Technologie Supérieure, Montreal, Canada
Rick Rabiser, CDL MEVSS, ISSE, Johannes Kepler University, Linz, Austria

Software Product Line (SPL) engineering has emerged to provide the means to efficiently model, produce, and maintain multiple similar software variants, exploiting their common properties, and managing their variabilities (differences). With over two decades of existence, the community of SPL researchers and practitioners is thriving as can be attested by the extensive research output and the numerous successful industrial projects. Education has a key role to support the next generation of practitioners to build highly complex, variability-intensive systems. Yet, it is unclear how the concepts of variability and SPLs are taught, what are the possible missing gaps and difficulties faced, what are the benefits, or what is the material available. Also, it remains unclear whether scholars teach what is actually needed by industry. In this article we report on three initiatives we have conducted with scholars, educators, industry practitioners, and students to further understand the connection between SPLs and education, i.e., an online survey on teaching SPLs we performed with 35 scholars, another survey on learning SPLs we conducted with 25 students, as well as two workshops held at the International Software Product Line Conference in 2014 and 2015 with both researchers and industry practitioners participating. We build upon the two surveys and the workshops to derive recommendations for educators to continue improving the state of practice of teaching SPLs, aimed at both individual educators as well as the wider community.

CCS Concepts: •**General and reference** → **Surveys and overviews;** •**Social and professional topics** → **Computing education;** •**Applied computing** → **Education;** •**Software and its engineering** → **Software product lines;**

Additional Key Words and Phrases: Software Product Lines, Variability Modeling, Software Product Line Teaching, Software Engineering Teaching

1. INTRODUCTION

The need to develop multiple variants of a system or artifact – being it a software-intensive system, an operating system, a user interface, a software architecture, a design model, or a test suite – is becoming more and more prevalent in numerous industries and domains. The ad-hoc development of multiple similar variants has a significant cost and dramatically increases time-to-market. As a result, practitioners increasingly need to adopt appropriate techniques for modeling and managing common properties, as well as variabilities (differences) of a system [Weiss and Lai 1999; Czarnecki and Eisenecker 2000; Clements and Northrop 2001; Gomaa 2005; Pohl et al. 2005; van der Linden et al. 2007; Apel et al. 2013].

In response, the field of *Software Product Line* (SPL) engineering [Weiss and Lai 1999; Clements and Northrop 2001; Pohl et al. 2005] has emerged to provide efficient methods, techniques, languages, and abstractions for practitioners in charge of producing and maintaining a family of related systems. With around two decades of existence, SPL engineering is now well-established in research and industry [Berger et al. 2013a]. The body of knowledge collected and organized by the SPL research community is still growing. Also, the scope of the community continuously broadens to other areas such as software ecosystems [Bosch and Bosch-Sijtsema 2010] and dynamic adaptive systems [Morin et al. 2009; Bencomo et al. 2012; Hinchey et al. 2012]. The long-term goal of the community is to provide systematic engineering methods, languages, and tools to assist practitioners in building well-structured and customizable systems.

However, without any effort for disseminating this knowledge, engineers of tomorrow are unlikely to be aware of the issues faced when engineering SPLs (or configurable systems) – up to the point they will not recognize this kind of system. In turn, they will not use appropriate techniques and face problems such as scalability, that the SPL community perhaps already studied or solved.

[1]The present submission constitutes a substantial extension of the paper: Mathieu Acher, Roberto E. Lopez-Herrejon, Rick Rabiser: "A Survey on Teaching of Software Product Lines". In Proceedings of the Eight International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2014). pp. 3 – 10. ACM, 2014.

We believe *education* has a key role to play. The teaching of SPLs can enable the next generation of engineers to build highly complex, adaptive, and configurable software systems that have been more frequently reported in industry [Berger et al. 2013a; Dubinsky et al. 2013; Vierhauser et al. 2014] or observed in the open source community [Berger et al. 2013b; Liebig et al. 2010]. Also, research can benefit from teaching: students can be involved in controlled experiments and researchers involved in teaching can identify potential missing gaps of SPL engineering and modeling tools and techniques.

Teaching SPLs is challenging. Software engineering itself is a relatively young discipline: it is still challenging to find good teaching methods and the correct place in a rather large and evolving curriculum [Ghezzi and Mandrioli 2005]. Moreover, SPL engineering encompasses a variety of topics, including requirements analysis, design, implementation, testing, modeling, and evolution. Another related challenge is to prepare teaching material – based on existing books, tools, and research papers – suitable for attracting and supporting students. Similar challenges have been reported in other software engineering communities, e.g., global software engineering [Beecham et al. 2017] and model-driven engineering [Kuzniarz and Martins 2016]. To the best of our knowledge, however, there is no published research work (except our own work [Acher et al. 2014a]) related to the teaching of SPLs.

Besides, many empirical studies and systematic reviews of both research and industrial practices of SPLs have been performed [Berger et al. 2013a; Benavides et al. 2010; Abbasi et al. 2013; Alves et al. 2010; Chen and Babar 2011; Czarnecki et al. 2012; Rabiser et al. 2010; Hubaux et al. 2010; Holl et al. 2012; Engström and Runeson 2011; da Mota Silveira Neto et al. 2011; Johansen et al. 2011] (to name but a few), but none of them addresses the teaching aspect of SPL engineering.

Currently, it is thus unclear how SPLs are taught, what are the possible gaps and difficulties faced, what are the benefits, or what is the material available. As a result there is an important educational gap between all stakeholders of the SPL community – researchers, educators, students, and industry practitioners. There is a risk, for instance, that educators neglect industry needs and inadequately train students (the future employees). Without any effort for disseminating SPL knowledge, engineers of tomorrow (students) will most likely not adopt state-of-the-art techniques developed by researchers and the 'wheel' will be constantly re-invented.

In this article we report on three initiatives we have conducted with scholars, educators, industry practitioners, and students to further understand the connection between SPLs and education (see Figure 1). Firstly, we conducted an online survey with the purpose of capturing a snapshot of the state of teaching in the SPL community [Acher et al. 2014a]. Our goal was to identify common threads, interests, and problems and build upon them to further understand and hopefully strengthen this important need in our community. 35 scholars provided us with detailed information about how they teach their SPL courses, what challenges they have and what they think about the impact of teaching. We published an overview of the results of this survey in a workshop paper [Acher et al. 2014a]. Secondly, we conducted another survey, based on the first one with scholars, but this time with students to find out about their experiences regarding learning about SPLs. 25 students provided detailed information about what and how they learned about SPLs. Thirdly, we organized two workshops on SPL teaching at the 18th and 19th International Software Product Line Conference (SPLC 2014 and 2015) [Acher et al. 2014b; Acher et al. 2015]. Experience reports were presented and discussed. We also invited experienced SPL practitioners as panelists for discussing SPL teaching and industry needs. We further discussed the creation of a repository to collect materials for teaching SPL engineering, which has now been made alive (http://teaching.variability.io) and already hosts contents from several sources.

This article reports, analyzes, and discusses the results of the two surveys (including a comparison of the results) and the discussions that took place at the SPL teaching workshops, with a particular focus on industry needs. We build upon all this material to sketch concrete recommendations for educators to continue improving the state of practice of SPL teaching. The intended audience of this article are: (1) educators, researchers, or industry practitioners interested in SPLs and variability modeling; (2) educators in a broad sense that aim to build software engineering curricula.

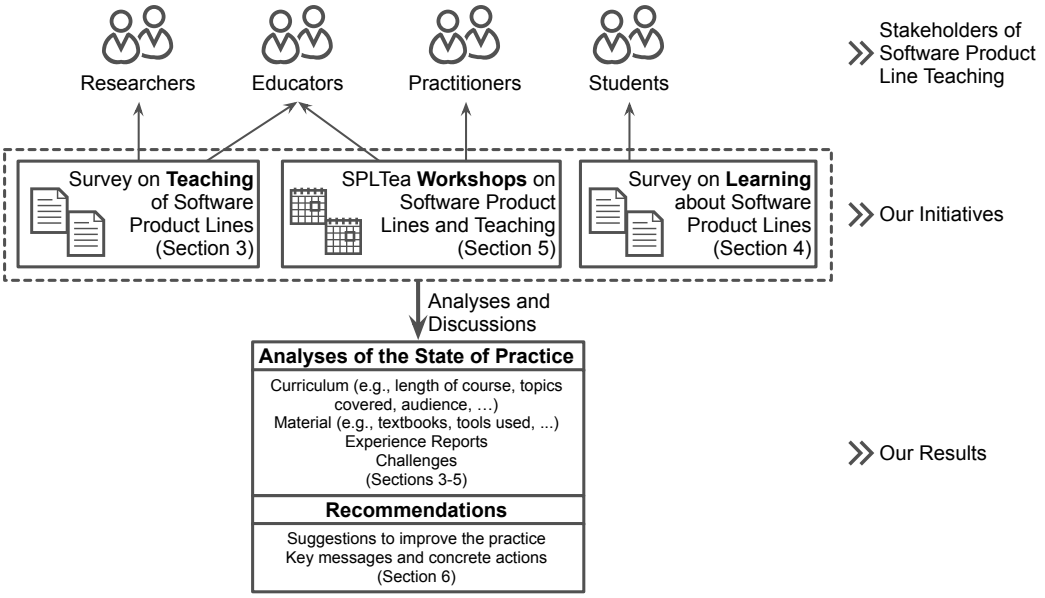

Fig. 1: We conducted three initiatives to gather evidence from researchers, educators, students, and practitioners with regard to the state of practice and challenges of software product line teaching.

2. OVERVIEW AND RESEARCH APPROACH

Figure 1 gives an overview of our research method as well as the structure of this article. To *gather evidence on the state of practice and challenges of software product line teaching* we decided to involve all relevant stakeholders, i.e., researchers, educators, students, and practitioners via three initiatives.

Firstly, we conducted a *survey on teaching software product lines among researchers and educators*. The goal of this first survey was to qualitatively and quantitatively analyze the state of practice of SPL teaching by involving researchers who either have held courses dedicated to SPLs or courses including SPLs as a topic. Based on our own experiences in teaching SPLs, we created a questionnaire and iteratively refined it based on discussions among the authors of this article as well as through feedback we gathered by informally presenting the questionnaire to colleagues at the Software Product Line Conference (SPLC) 2013 in Tokyo. More specifically, we asked peers to complete the questionnaire and comment on any issues that might occur to them. At least one of us was available and collected the feedback in a central document for later discussion among all authors.

The resulting questionnaire (the full list of questions can be found in the appendix of this article) comprised 13 questions intended to collect quantitative results (demographic data) about the context of teaching SPLs – i.e., country, institution, department, experience of the instructors, targeted audience, length of the course, primary literature used, tools used, and parts of the SPL lifecycle covered – as well as five open questions to encourage qualitative and constructive remarks about the experience and challenges of SPL teaching.

We created a list of contacts to whom we sent our survey. We created this list based on our own contact networks, by searching for SPL courses online, and by reviewing the list of authors and co-authors of papers accepted at the Software Product Line Conference (SPLC) series as well as the Variability Modelling for Software-intensive Systems (VaMoS) workshop series from 2009-2013 (the teaching survey was conducted at the end of 2013). Details on the design as well as the results

of the first survey on SPL teaching among scholars are presented in Section 3 and have also been summarized in a workshop paper [Acher et al. 2014a].

Secondly, based on the experiences we made with our first survey, we designed and performed a second *survey to analyze the state of practice of SPL learning, i.e., to include the student perspective*. We created a first draft of the questionnaire and informally presented it to students at courses in Oxford and JKU Linz. Specifically, we asked students to complete the questionnaire and comment on any issues that might occur to them. Based on students' remarks, we refined some of the questions, mainly to make them more precise and understandable for students.

The resulting questionnaire (the full list of questions can be found in the appendix of this article) comprised six questions intended to collect quantitative results (demographic data) about the context of learning SPLs – i.e., location, name of university or college; program (e.g., Computer Science); highest degree (e.g., Master, Bachelor); students' background knowledge; and current job – as well as eight open questions to encourage qualitative and constructive remarks about the experience of SPL learning.

We directly sent the survey to our own students (three courses held; one in Rennes, one in Oxford, and one in Linz). We also contacted the respondents of the first survey asking them to forward the survey to their students. Details on the design as well as the results of this second survey on SPL learning among students are presented in Section 4.

In both surveys, we analyzed results in a similar manner:

- We decided to only take complete responses (answers to all survey questions provided) into account in our quantitative analyses but also double checked partial responses to find additional qualitative suggestions, i.e., we used all answers to qualitative questions, even from only partially completed questionnaires.
- All survey responses were discussed among the three authors of this article and possible interpretations were collected in a central/shared document. We considered challenges that were most frequently mentioned by survey respondents as most important and thus worthy to report about/discuss them. For conflicting interpretations, we discussed them among all three authors to eventually reach mutual agreement.
- For the quantitative questions, we defined categories to present the results in an organized manner, e.g., the category *course length, audience, and practical time* groups the answers to five questions on the context (self-contained course vs. part of another course), the audience, the total lecture time, the total practical time, and the length of the course.
- For all survey questions we first present the responses (in a summarizing manner) before adding our personal interpretations.

Thirdly, based on the two surveys that involved scholars and students, we organized *workshops to collect additional feedback from the SPL research community as well as from industrial practitioners* about what we learned in the surveys. Via an open call for papers at the 18th International *Software Product Line Conference (SPLC)* in 2014 we invited submissions reporting research on SPL teaching, discussing experiences with SPL teaching, as well as position and vision papers [Acher et al. 2014b]. At SPLC 2015, we again organized the workshop to further discuss important issues [Acher et al. 2015]. This time we put a special focus on the design and population of an open repository (http://teaching.variability.io) of resources dedicated to SPL teaching. During the workshops all participating authors took electronic notes. After the workshops we exchanged (collected in a central document) and then discussed (in a conference call) our notes in the light of the results of our two surveys, to categorize/organize them and derive general conclusions. Conflicting notes were discussed among all three authors during the call to eventually reach mutual agreement of our interpretations. In Section 5 we summarize the results of the SPL teaching (SPLTea) workshops held at SPLC 2014 and SPLC 2015 including the discussions of an *industry panel*.

For each initiative (teaching survey, learning survey, and the two workshops) we systematically discuss threats to validity of the presented results.

Section 6 builds upon the results of the two surveys and the two workshops as well as our discussions and interpretations. It highlights important points that need to be addressed in the future. Specifically we provide concrete recommendations for SPL educators as well as the wider community of computing education.

3. SURVEY ON TEACHING OF SPLS

The goal of our first survey was to qualitatively and quantitatively analyze the state of practice of SPL teaching by involving researchers who either have held courses dedicated to SPLs or courses including SPLs as a topic. Please note that this section is based on our earlier publication at the VaMoS Workshop 2014 [Acher et al. 2014a].

3.1. Research Method

When selecting survey participants, we considered that a public dissemination of the survey (e.g., on general mailing lists) can reach a large audience but also can have counter effects. Survey participants should be aware of SPL engineering and have experience in teaching. Public dissemination can easily lead to many irrelevant or incomplete answers, because most participants then might not be concerned with SPLs or teaching (of SPLs). Furthermore, public dissemination might not reach many relevant respondents, i.e., those that are in fact teaching SPLs.

Therefore, we decided to directly contact potential participants by relying on our own networks and based on a Web search on SPL courses (cf. Table IV for some examples). First, we selected researchers from the SPL community we already know are actually teaching on SPLs. Second, we extended this initial list with the teachers of courses we found online, that were not already on this list. These two steps resulted in a list of 41 people. As a third step, we reviewed the list of authors of papers accepted at the Software Product Line Conference (SPLC) series as well as the Variability Modelling for Software-intensive Systems (VaMoS) workshop series from 2009-2013 (the teaching survey was conducted at the end of 2013) and selected people not yet on our list. This action was particularly useful because in the SPL community researchers very often also teach. As a result, we could augment our original list with 51 more scholars.

Our final list thus had 92 contacts for our survey. Because of the fact that we manually created this list, we were confident it contained people that actually are teaching SPLs. When creating our list, however, we did not distinguish whether people teach SPLs as a self-contained course or as part of another lecture like software engineering. Additionally, when sending out the survey, we invited the 92 people we contacted to forward the survey to their own contacts, who also teach SPLs.

We used the tool SurveyGizmo to set up our survey as an on-line questionnaire (see http://www.surveygizmo.com/s3/1342346/Teaching-Software-Product-Lines and the appendix of this article). Before we sent out the survey, however, we collected feedback from our peers. More specifically, we informally presented it to colleagues with experience in SPL teaching, e.g., at the Software Product Line Conference in Tokyo in August 2013. More specifically, we asked our colleagues to complete the questionnaire and comment on any issues that might occur to them. Based on colleagues' remarks, we – together, during a dedicated conference call – refined some of the questions, mainly to make them more precise.

The resulting final survey contained 13 questions that intend to gather quantitative results (demographic data) regarding the context of SPL teaching, i.e., country, institution, department, experience of the instructors, targeted audience, length of the course, primary literature used, tools used, and the parts of lifecycle of SPLs covered. Each of these 13 questions comes with a set of possible answers to select from. Most of them also come with an 'other' option together with a text field allowing participants to specify an additional or different answer themselves. Additionally, our survey comprised five open questions encouraging qualitative and constructive remarks regarding the experience of SPL teaching. Specifically, we asked survey participants what the most challenging part of teaching SPLs is, what suggestions they have to improve the state of teaching SPLs, how SPL teaching can impact SPL research and industrial practice, and if they have any other comments, concerns, or remarks relevant to teaching SPLs or the survey in general.

Distributing the survey to the 92 contacts via mail led to 35 complete responses (with all survey questions completed) and 15 partial ones (with only some survey questions completed). We decided to only take complete responses into account in our quantitative analyses but double checked partial responses to find additional qualitative suggestions, i.e., we used all answers to qualitative questions, even from only partially completed questionnaires.

3.2. Results

In the following we summarize the results we collected with our survey and highlight the most important findings. Regarding the 13 quantitative questions of our survey, we defined five categories to present the results in an organized manner:

- Category ***'respondents and their institutions'*** contains the answers to the five questions on the name and country of the respondents' institution, the type of institution, the department, and the experience of respondents in SPL research and teaching.
- Category ***'primary literature used'*** presents respondents' answers on what primary literature they use in their course
- Category ***'tools used'*** presents what tools respondents mentioned.
- Category ***'course length, audience, and practical time'*** comprises the answers to five questions on the context (self-contained course vs. part of another course), the audience, the total lecture time, the total practical time, and the length of the course.
- Category ***'parts of the SPL lifecycle covered'*** reports what parts of the SPL lifecycle respondents said they cover in their courses.

3.2.1. Respondents and their institutions. The 15 different countries the 35 respondents of our survey teach in are depicted in Figure 2. Not surprisingly, these 15 countries are also (among the) key players in SPL research. Respondents' average experience in SPL research is over ten years (median: 8; min: 0; max: 37) while their average experience in SPL teaching is over six years (median: 5; min: 1; max: 20).

Interestingly, almost half of the respondents have the same experience in SPL research and in SPL teaching. One possible explanation for this fact is that SPL teaching is often performed concurrently with research. While this allows SPL teachers to teach about current trends and technologies, teaching in the SPL community thus might not always be the result of mature research experience.

With regard to the types of institutions: Most (16) are research-focused or (9) teaching-focused (colleges), seven focus on both, and three are pure industrial institutions. SPL topics are mainly taught at computer science (13 responses) or software engineering departments (10 responses) and less in information technology departments (2 responses). Other mentioned departments (10 responses) were: software science and technologies, computer science and computer engineering, industrial and management systems engineering, information systems, mathematics and informatics, systems and computing engineering, informatics, as well as computing and information systems.

About half (17) of survey respondents teach full and self-contained SPL courses. The other half (18) teach about SPL topics as part of their other courses, e.g., in software engineering (8), requirements engineering (4), automated software design (1), principles of software construction (1), domain engineering (1), software architecture (1), and factory development of software (1). Given the relative novelty of the SPL topic, we do not find it surprising that over half of respondents teach SPL topics as part of their other courses.

3.2.2. Primary literature used. The key primary literature (text books) used in teaching SPLs is shown in Figure 3 in alphabetical order: Apel et al. [Apel et al. 2013], Clements and Northrop [Clements and Northrop 2001], Czarnecki and Eisenecker [Czarnecki and Eisenecker 2000], Gomaa [Gomaa 2005], Pohl et al. [Pohl et al. 2005], van der Linden et al. [van der Linden et al. 2007], and Weiss and Lai [Weiss and Lai 1999].

Most (29) respondents of our survey use these books, many (26) use research papers, and only about a third (13) use case studies. Six respondents mentioned case studies from van der Linden et

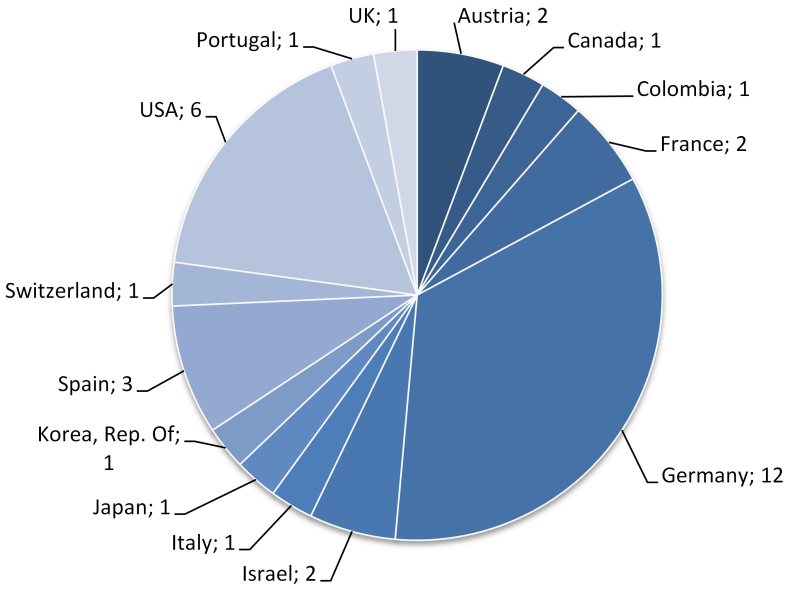

Fig. 2: Respondents' locations.

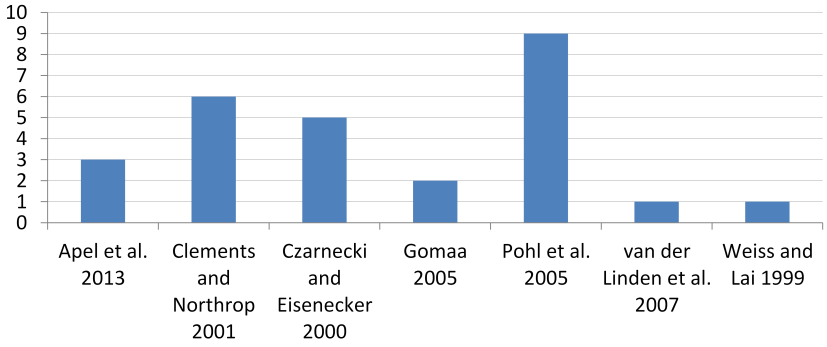

Fig. 3: Primary literature (books) used in teaching SPLs.

al. 2007 [van der Linden et al. 2007] (2); BigLever case studies (1); Renault, STAGO, LINUX, and SPLOT models (1); and their own case studies (2). Several research papers were also listed by some respondents; however, only one was mentioned more than once, i.e., the original FODA report by Kang et al. [Kang et al. 1990]. Papers generally seem to be selected based on teachers' personal preferences and topics they cover.

Even though using textbooks and maybe some research papers for teaching is pretty much standard in computer science, using case studies is equally essential, particularly regarding teaching SPLs. The fact that only about a third of our survey respondents use case studies to teach SPLs should thus be further investigated (cf. Section 6).

3.2.3. Tools used. Successfully applying SPL techniques required adequate tools. We thus were interested in finding out what tools are used in the taught courses: Most (27) respondents said they

Table I: SPL tools used for teaching SPLs.

SPL Tool	**# uses**
None	8
FeatureIDE	6
BigLever's GEARS	5
FeatureHouse	4
pure system's pure::variants	4
AHEAD	2
CIDE	2
CVL	2
Feature Modelling Plug-in (FMP)	2
DOPLER	1
EasyProducer	1
FaMa	1
Familiar	1
FeatureMapper	1
Munge	1
SPLAR	1
SPLOT	1
VARIAMOS	1
Varmod	1
C++, Metaprogramming, Software Generators, Xtext, MPS	1
Different tools for creating feature diagrams and UML-based models	1
Feature modeling UML tools extended for SPL	1
Haskell-Embedded Variation DSL	1
Velocity Template Engine	1
AspectJ	1
Own research prototypes	1

use SPL tools while only about a quarter (8) said they don't use any tools. Table I shows an overview of the mentioned tools.

The survey on variability modeling in practice by Berger et al. [Berger et al. 2013a] shows a similar trend: industry-strength, commercial variability management tools such as BigLever's GEARS and Pure Systems's pure::variants are used, but prototypes are more commonly used than commercial tools. Many research groups use their own research prototypes. The tool FeatureIDE is more widespread than other tools that can be considered research prototypes. Most mentioned tools, particularly the research tools, mainly focus on supporting feature-oriented software development and/or variability modeling.

3.2.4. Course length, audience, practical time. Regarding course length, most courses are held over a whole semester (26), followed by courses offered 'on demand' (e.g., depending on number of registrations) (3), week-long courses (3), courses over a quarter (1), 1-2 days seminars (1), and courses with a flexible schedule, i.e., held for one week (blocked) or not frequently but over a whole semester (1). The average course length for a course held for one semester is 24 hours the minimum length is 1 hour and the maximum length is 120 hours.

The target audience of SPL courses is depicted in Table II. It shows a specific trend: most courses are held for graduates and/or industry practitioners. The relation of practical time to overall course time (i.e., exercises, working on a concrete project) – 65% – also reflects this fact: almost two thirds of course time is thus spent in practical exercises and work. One respondent even specified 2.5 times more practical time than lecture time. However, for eight courses respondents said they allot no practical time at all.

Table II: Target audience of SPL courses named by survey respondents.

Target audience	**# responses**
Graduates	14
Undergraduates	6
Undergraduates and graduates	5
Graduates and industry	5
Undergraduates, graduates, and industry	3
Industry	2
Undergraduates and industry	0

Table III: Parts of the SPL development lifecycle covered in SPL courses.

Topic	**Department**			**Total**
	CS	SE	OT	
Modeling	9	9	13	30
Implementation	9	9	8	25
Requirements engineering	7	5	12	23
Processes	5	6	8	19
Maintenance & Evolution	9	3	5	17
Reverse engineering & SPL Adoption	7	2	1	10
Other	5	2	3	10
Testing	3	2	4	8

CS: Computer Science, SE: Software Engineering,
OT: Other departments.

3.2.5. Parts of the SPL lifecycle covered. The parts of the SPL development lifecycle that are covered in SPL courses as specified by survey respondents are depicted in Table III. Modeling, requirements engineering, and implementation are the main foci of teaching SPLs. Processes and Maintenance & Evolution are also essential. Reverse Engineering and Adoption and Testing are not as frequently covered, despite the extensive research existing and being conducted in both areas. Respondents also commented that they cover other parts of the SPL development lifecycle: business and organizational foundations (4), verification (2), architecture and design (2), decision models (1), measurement and analysis (2), tools (1), scoping (1).

Additionally, we divided topics by departments, i.e., computer science, software engineering and other, and for each topic highlight the department with the highest count. There is a clear and even split between computer science and other departments, i.e., each received the highest count in half of the topics. This may suggest a trend in the courses taught at computer science departments towards implementation, maintenance and evolution, reverse engineering and adoption, and other miscellaneous topics that may include more advanced research.

In contrast, courses offered at other departments lean more towards requirements engineering, modeling and processes. The only exceptions were modeling and implementation where computer science and software engineering programs were tied and closely followed by the other departments. This suggests that regardless of the department at which an SPL course is taught, all courses consider modeling and implementation an important part of the curriculum. It would be interesting to corroborate or disprove these trends and their potential causes in future work.

3.2.6. Challenges of teaching SPLs. The first open question of our survey was: *'What is the most challenging part for teaching SPLs? (e.g., administrative delays, acceptance in the curriculum, lack of material)'*. Only four respondents commented that they see no challenges, all others at least

mentioned one challenge. Analyzing and categorizing all comments resulted in the following key challenges regarding SPL teaching:

Lack of and availability of SPL tools. According to six respondents *'a key challenge is that many SPL tools are not freely available'*, e.g., under an open source license. The tools that are available also are often only poorly documented. Three respondents also pointed out that *'there is a lack of integrated tooling to show a complete tool chain'*. For instance, SPL tools often focus only on one part of the SPL development lifecycle such as feature modeling.

Lack of and availability of well-documented real-world examples and case studies suitable for teaching. Almost half of all respondents argued that the lack of and availability of examples and case studies is a key challenge for SPL teaching. Some commented they *'would require good textbook examples, i.e., small, plausible, yet non-trivial examples to entice students'*. Others argued that *'large-scale industrial case studies would be better'*. Respondents agreed that good, well-documented case studies and examples are hard to find despite the many available research papers and SPL textbooks describing case studies. Unfortunately, most available case studies and examples have not been developed and written for the purpose of being used in SPL teaching. Instead, they have been written to report interesting findings in the research community. The Arcade Game Maker Pedagogical Product Line [McGregor 2014] is an exception. Other material needs to be restructured, prepared, and organized for teaching.

Complexity of the subject and required background knowledge. Ten respondents commented that SPLs are a complex topic including many aspects (in addition to software engineering), which makes it hard to teach SPLs to computer science students and requires students to have strong background knowledge. One respondent summarized this challenge as follows: *'even software engineering can be hard to teach as developing large-scale systems does not connect to students hands-on experience of developing rather small solutions. Teaching SPLs means SE for many systems, this does even less relate to students' experiences'*. Other respondents also reported that students have difficulties understanding how to develop large-scale systems and usually have not much experience with big projects. Students require background knowledge in diverse areas such as model-driven development, software architectures, or constraint programming.

Acceptance on the curriculum and opening the mind of students for the topic. Two respondents commented that they had difficulties to get the SPL course accepted in their curricula. This could be related with the challenge regarding the complexity of the subject and the required previous knowledge. Two other respondents also commented that *'it can be quite tricky to open the mind of students for SPLs and make them interested'*.

3.2.7. How to improve the state of teaching SPLs. The next question we asked was: *'What would you suggest to improve the state of teaching SPLs?'*. We extracted and categorized the following suggestions from respondents' answers (30/35 provided at least a short comment):

Better tools for students. Eight respondents commented that teaching SPLs could be improved with 'better tools', i.e., *mature, stable, and 'industry-ready' tools students can use without extensive training*. Respondents also seem to want a standardized tool that is recognized by a large community and that is based on a common technological basis regarding, e.g., variability mechanisms and derivation technologies. Respondents, however, doubt research prototypes' usefulness for teaching SPLs. Research efforts such as the Common Variability Language (CVL) [CVL 2014] could be candidates to address this issue.

Improved textbook examples and case studies. Many respondents (14) see the need to improve (textbook) examples and (industrial) case studies for teaching SPLs. The key issue they identify here is that the existing material – textbook examples and case studies – is simply not designed specifically for students/teaching. Good examples and case studies should be a showcase for how and why SPLs are introduced in industry. Instead, unfortunately, they often make the subject look like a pure research endeavor. Also – as commented by one respondent – *'we must work on joining theory from the books with the tools used in practice'*, i.e., better map theoretical concepts and

practical solutions. This is particularly true for variability modeling, where theory and practice often differ [Berger et al. 2013a].

Broaden the focus of teaching SPLs. Three respondents suggested broadening the focus of SPL teaching, i.e., to 'grow out of the software engineering realm' and also teach and use similar approaches like component-based systems and/or service-oriented architectures to motivate SPLs. A specific suggestion by a respondent was to move away from feature models as putting the focus mainly on these models hinders progress. Instead the respondent suggested to encourage the use of domain-specific languages. In summary, respondents said that *feature models should be part of teaching SPLs but it is also important to show other approaches*.

Is SPL already mainstream? Two respondents argued that *'SPLs are already mainstream in industry and this information should be conveyed to students'*. However, industry does often not make use of very special techniques and approaches but instead develop their own custom solutions [Lettner et al. 2013] to support reuse and variability management. This means, when teaching SPLs one maybe should not teach too early-stage SPL approaches. On the other hand, early-stage approaches might one day become the mainstream. Regarding this issue, respondents commented that *'*a balance between industry-strength methods/tools and emerging research ideas should be found by instructors'.

Other ideas. An original idea by a respondent was to *'collect videos of experts that can be shown to students to motivate SPLs and explain key concepts and ideas'*. Another respondent suggested to *'incorporate the notions and terminology of SPLs more in other courses'*, which is something many scholars teaching SPLs presumably already do. Developing a standard curriculum and evaluation scheme for teaching SPLs, as suggested by another respondent, would also be a very important step to make SPLs really (even more) mainstream.

3.2.8. Impact of teaching on research. We also asked *'How can teaching improve/impact research on SPLs?'*. We received comments from 30 of 35 respondents and grouped them as follows.

Student participation in research evaluations. 12 survey respondents commented students are useful participants in evaluating respondents' research, either as subjects in experiments or in case studies. However, two other respondents commented that *'this can be a tricky issue because most likely only small, (quasi-)experiments can typically be done'* – depending on factors such as class size, course duration, experience level of students. Such small experiments typically do not provide robust empirical evidence. Overall, however, performing experiments with students as subjects is a good experience for both students and scholars. Even if the results are not that useful or directly usable for a research paper, they can provide preliminary evidence and direct further studies. We conclude that students can be very useful in performing pilot studies.

Feedback on and discussion about tools, examples, and case studies. Six respondents also said that *'students are a very useful audience when it comes to simply discussing SPL research results and developed tools'*. Students can provide useful feedback to researchers regarding the improvement of their research work (even if it only means improving the presentation of this work). Also, students can help with tool development and in elaborating and trying out examples and case studies.

Finding research personnel. Eight respondents commented that they think teaching SPLs is useful to find personnel, i.e., both research students or candidates for Master and PhD theses can be identified. Also, the general benefit of attracting students to the SPL community must not be underestimated, no matter whether students end up in industry or research.

Connecting with industry. Six respondents also said that *'some students in SPL courses either are already working in industry or will be working in industry soon'*. This helps to trigger future industry-academia collaborations or simply to increase awareness in industry.

Discussion of open research issues with students. Two respondents commented that they often discuss research issues with their students because *'getting a different view is always helpful to drive research, even if its 'only' from a student'*. Another respondent commented that *'missing teaching material is also sometimes an indicator for missing research'*, i.e., research topics that should be (further) pursued.

3.2.9. Impact of teaching on industrial practice. We then asked *'How can teaching SPLs impact industrial practice of SPL engineering?'*. 31 of 35 respondents provided comments, all rather similar or overlapping. Thus, here we do not categorize respondents' comments.

Over a fifth of respondents said that *'teaching SPLs makes students aware of the topic and students eventually end up in industry or even are already working in industry'*. Thus, teaching SPLs increases awareness of SPLs in industry and trains future practitioners. Teaching SPLs can even open the door to companies, which have not been previously exposed to SPLs.

13 respondents even argued that teaching SPLs is already a preparatory step for introducing SPL engineering in industry, i.e., *'students will be the future drivers of the adoption of SPL engineering in industry'*. Some students are already working in industry while they attend SPL courses and thus they might trigger a paradigm shift in their company. Furthermore, two respondents commented that teaching SPLs can be the start of industry-academia collaboration projects.

3.2.10. Other comments. Eventually, we asked participants for any other comments and concerns regarding teaching SPLs or the survey in general. We received comments from less than a third of respondents. We only discuss the ones that were not already covered above.

For instance, one respondent reported that they teach SPLs in combination with model-driven development and find the interplay of both approaches very interesting, also from a teaching perspective. We think there are indeed many commonalities between these two research areas.

Another survey respondent wrote *'having a common, clearly defined basis of terminology and concepts that is taught at the majority of the institutions would help a lot'*. Three respondents similarly commented that real-world SPL engineering is very different from research in many aspects. For instance, while feature models are (on of the) *the* key topic(s) in research, in practice they often are just one technique useful for some roles. Industry often manages variability without dedicated models, e.g., using spreadsheets, and they are still successful with their product lines. This is a fact that should be reflected in teaching.

We finally received one comment that *our survey was not a perfect fit for SPL consultants and trainers in industry*. We agree with that and tried to address this issue by explicitly involving industry practitioners in a follow-up workshop (cf. Section 5.3).

3.3. Threats to Validity

Our survey exhibits a number of threats to validity.

The survey was designed by just three researchers (the authors of this paper), based on their experience in research and teaching SPLs. To mitigate this threat, we asked our peers for feedback and refined the survey incorporating their feedback. Nevertheless, others might have asked different questions and, particularly, might have provided different possible answers for questions.

Another threat regarding the internal validity of our results is that survey respondents misunderstood questions. We addressed this threat by test-driving the survey questionnaire with several peers with experience in teaching SPLs. Further, the open questions at the end of our survey allowed participants to raise any concerns and ask for clarification. Two survey participants even contacted us directly (via mail) before completing the survey asking some clarification questions. We discussed how to respond to them to not bias them with our response but decided to help them.

A third threat is that some people might not have had the required knowledge to answer all of our questions. For example, some of the authors of SPLC papers we contacted might not also teach SPL-related courses and still answer the survey. While it is common practice in the SPL community that researchers also teach, not every researcher teaches or has experience in (SPL) teaching. In addition to the assumption that someone who does not teach will not also complete the survey (remember: we only counted complete responses), we also allowed respondents to write comments indicating they do not teach SPLs.

An external threat to the validity of our research is that, perhaps, we did not contact a representative pool of SPL educators. For instance, one can easily see that our survey is dominated by responses from Germany. However, this just reflects the (current) situation in the SPL community.

At the 2016 SPLC conference in Beijing, submission and paper acceptance statistics as well as participant statistics presented in the welcome session showed that Germany has by far the most researchers working on SPL topics. In addition to selecting teachers we know, we also performed a web search for SPL courses and we reviewed the list of authors of papers published at the main SPL events SPLC and VaMoS. Not surprisingly, many are from Germany.

Furthermore, as also commented by one respondent, the teaching survey was not a perfect fit for SPL consultants and trainers in industry. To address this threat, we explicitly involved industry practitioners in the SPLTea workshops described below (see Section 5.3). The fact that for some open questions less than a third of the survey respondents replied is another reason why we back up the results of our survey with the results of discussions at these workshops.

The rather low number of (complete) responses in our survey (35) also might decrease the usefulness of any quantitative analysis. While we indeed present some quantitative (demographic) data, we decided to focus also a lot on qualitative results. Our possibly wrong interpretation of answers given to open questions is thus another important threat. We are confident, however, that we could capture the essence of respondents' answers in our interpretations, particularly, as for most answers we could find more than one similar instance, often even more than five.

4. SURVEY ON LEARNING ABOUT SPLS

Based on our first survey, the goal of our second survey was to qualitatively and quantitatively analyze the state of practice of SPL learning, i.e., to include the student perspective.

4.1. Research Method

We directly sent the survey to our own students (three courses held; one in Rennes, one in Oxford, and one at JKU Linz). We also contacted the 92 respondents of the first survey asking them to forward the survey to their students. Again we did not consider if the contacts teach SPLs as a self-contained course or as part of another lecture such as courses on Software Engineering in general. However, one of the questions of the survey can be used to explicitly indicate this information, allowing us to treat answers accordingly.

We again set up the survey as an on-line questionnaire (it can be found at http://www.surveygizmo.com/s3/1741319/Learning-Software-Product-Lines and in the appendix of this article). Before sending the link to our students and contacts, we created a first draft of the questionnaire and informally presented it to students at courses in Oxford and JKU Linz. More specifically, we let students complete the questionnaire and comment on any issue that might occur to them. Based on students' remarks, we – based on discussion among all three authors – refined some of the questions, mainly to make them more precise and understandable for students.

The resulting questionnaire (the full list of questions can be found in the appendix of this article) comprised six questions intended to collect quantitative results (demographic data) about the context of learning SPLs, i.e., location, name of university or college; program (e.g., Computer Science); highest degree (e.g., Master, Bachelor); availability of at least basic knowledge in Software Engineering, Requirements Engineering, Software Architecture and Design, Reverse Engineering and Software Maintenance, Software Processes; current job (only student or also working in industry; if yes, in which area). For each quantitative question, we provided a set of possible answers to select from and – for most of these questions – we allowed survey participants to select 'other' and specify a more accurate or additional answer themselves. The questionnaire, however, also comprised eight open questions to encourage more qualitative and constructive remarks about the experience of SPL learning, which was the main focus of this survey. More specifically, we asked participants:

- What did you expect from this course before it started?
- What did you actually learn from this course?
- What did you not learn and/or would have wanted to hear more about?
- Please define in your own words what a Software Product Line is.
- What do you think is the most challenging part of learning about Software Product Line?

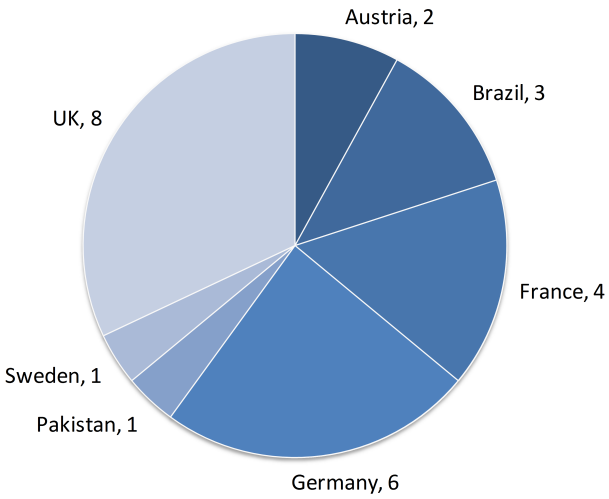

Fig. 4: Students' locations.

- Suggestions to improve the state of the teaching of Software Product Lines?
- How can the teaching of Software Product Lines impact industrial practice?
- Any other comments (for example: what did you like the most, what did you like the least, why?)

We received 25 complete responses[2] (with all questions of the survey completed) and 37 partial responses (only few questions completed). We decided to only take complete responses into account in our quantitative analyses but double checked partial responses to find additional comments for our qualitative analyses, i.e., we used all answers to qualitative questions, even from only partially completed questionnaires.

For the six quantitative questions, we defined two categories to present the results in an organized manner. More specifically, we report on *basic statistics* (countries, programs, degrees, and job) and *background knowledge* of students in basic fields (Software Engineering, Requirements Engineering, Software Architecture and Design, Reverse Engineering and Software Maintenance, Software Processes). For the eight qualitative questions we discuss the answers to every question in detail.

4.2. Results

In this section we summarize the results gathered in our SPL learning survey and highlight the important findings.

4.2.1. Basic statistics. Figure 4 shows the seven different countries represented by our 25 student survey respondents, i.e., the locations of the universities or colleges the students attended a course on SPLs (not the students' origin countries, which are not relevant for the purpose of this survey). Our results have a strong focus on the situation in European countries and Brazil. We still think its worthwhile reporting these results, but in our interpretations of open questions we took this fact into account. For instance, answers on the question about the impact of teaching SPLs on industrial practices of course depend a lot on the industry landscape in a particular country.

Of the 25 respondents, 14 participate in a computer science and 11 in a software engineering program at their universities. 12 have a Master's degree already, while 11 have a Bachelor's degree, and there are already 2 with a PhD but from different disciplines. 9 students are full-time students, while 16 are also working in industry in parallel to their studies. Of these 16, 13 have

[2]Including responses from our own courses in Austria, UK, and France.

a Master's degree and 3 a Bachelor's degree. We also asked students in which area they work in. For this question, we received quite diverse replies, i.e., financial services (3), industrial automation (2), software engineering or programming (2), consulting for software reuse adoption, gaming industry, government, insurance, marketing, payroll applications, PIM software development, general software development, packaging industry, software maintenance and security, and web application development.

Table IV provides an overview of the particular courses the 25 students participated in (eight different courses). While two courses focus on (systematic) software reuse in general, one focuses on model-driven engineering, and one course has a particular focus towards programming, all courses cover the key processes in the SPL engineering lifecycle, i.e., domain and application engineering including, for instance, variability management and modeling, variability implementation, as well as product line testing. We thus think the heterogeneity of courses is not a big influence factor on the interpretation of our results.

4.2.2. Students' background knowledge. As depicted in Table V, all 25 respondents commented that they have at least basic knowledge in software engineering. 21 replied that they have knowledge in software architecture and design. 20 respondents mentioned they knew about software processes and 19 have knowledge about requirements engineering. The least frequent background knowledge was reverse engineering and software maintenance with 11 respondents. Nonetheless, these numbers indicate a strong background in the software engineering core and most salient topics, which is indeed a perceived prerequisite to fully appreciate SPLs.

4.2.3. Expectations. We asked what the expectations of students were before the SPL course started (or before the SPL topic was addressed in another course they attended). We received some *useless replies* such as *'I learned to expect nothing from a university course'* or *'get credit'* but also some *interesting ones*. Nine students replied that they *'expected to learn the basics of software reuse and SPLs, particularly their advantages and disadvantages'*. Three other students (both having an industry job) replied that they not only expected to learn these basics but especially also *'how SPLs work or can be implemented in practice'*. We think this is a key message for teaching SPLs: *students with industrial experience expect to learn not only theories but also how to apply them in practice*.

While this is certainly true for most subjects in computer science, we think SPL knowledge is particularly hard to teach without practical examples. This is confirmed by our teaching survey as well as by other replies of students on their expectations, i.e., that they wanted to learn:

- *'managing a complex software system'*.
- *'more about product management and testing'*.
- *'handling the process of developing similar software'*.
- *'different approaches how SPLs could be implemented'*.

In summary, comments show that students expect a course on SPLs to teach them both, the basics of SPLs as well as how to apply and implement SPLs in practice.

4.2.4. Perceived learning effects. In contrast to the question on what they expected to learn, we asked students what they actually learned from the SPL course. Eight students replied that besides the basics and theory of SPLs they also learned how to apply SPL engineering in practice based on presented case studies and implementation examples. For instance, one reply was: *'I learned several approaches how SPLs could be implemented, the advantages & disadvantages of them and got live experience of how much programming effort each approach takes'*. Only two students replied that they did only learn a little about applying SPL engineering in practice, but instead mainly heard about theory. All students replied that they learned the basics of SPL engineering and the involved processes (application and domain engineering). Three students mentioned variability modeling/feature modeling as a key topic they learned about. Some students mentioned concrete approaches such as aspect orientation (4), feature orientation (3), and delta-oriented programming (1).

Table IV: Information about the courses students responding to our survey participated in.

Course	**Audience**	**Main focus**	**Country: num respondents**
Product Line Engineering (https://lss.jku.at/studienhandbuch/30223)	CS Master's students	covers the full SPL engineering lifecycle	Austria (JKU Linz): 2
Systematic Software Reuse (http://homes.dcc.ufba.br/~esa/disciplinas/ssr20152.html)	CS Master's and PhD students	provides an overview of systematic software reuse including SPLs (covering full SPL engineering lifecycle)	Brazil (UFBA Salvador): 3
Model-Driven Engineering (http://teaching.variability.io/rennes2015-MDE.html)	MSc students	focuses on (model-driven) variability management as well as reverse engineering and testing of product lines	France (Universite de Rennes 1): 4
Software Produktlinien - Konzepte, Analysen und Implementierung (http://www.es.tu-darmstadt.de/lehre/aktuelle-veranstaltungen/spl-v/)	CS Bachelor's and Master's students	covers the full SPL engineering lifecycle	Germany (HS Darmstadt): 3
Software Product Lines: Concepts & Implementation (https://www.tu-braunschweig.de/isf/teaching/2016s/spl)	Master's students in CS, Business Informatics, and Information Systems	covers the full SPL engineering lifecycle but with a focus on programming	Germany (TU Braunschweig): 3
Software Reuse (no URL)	CS Master's students	provides an overview of systematic software reuse including SPLs	Pakistan (Lahore Univ.): 1
Software Product Line Engineering (http://utbildning.gu.se/education/courses-and-programmes/Course_detail?courseId=DIT275)	CS Master's students	covers the full SPL engineering lifecycle	Sweden (Univ. of Gothenburg): 1
Software Product Lines (no URL)	Part-time MSc students from industry	covers the full SPL engineering lifecycle	UK (University of Oxford): 8

Table V: Students' experience in different areas of knowledge relevant for SPLs.

Knowledge area	**# respondents with experience**
Software Engineering	25
Software Architecture & Design	21
Software Processes	20
Requirements Engineering	19
Reverse Engineering & Software Maintenance	11

What can be learned from students' replies is that SPL courses seem to focus on both, the basics and theory of SPLs as well as applying SPL engineering in practice (taught via case studies). When comparing the expectations of students with what they actually learned no student learned less than expected, most learned more. For instance, one student also commented that the course helped him to *'sense the difficulty of designing software that is maintainable and able to evolve from the very beginning'*.

4.2.5. Missed student interests. We also asked students what they did not learn but would have wanted to learn. 5 of the 16 students working in industry wanted to be taught *more practical examples and tools or more about recent case studies or experience reports from SPLs in practice*. Two students explicitly mentioned that they wanted but did not learn about the *barriers and challenges for adopting SPL concepts in a real software organization and how to introduce the SPL culture in a company*. Another student explicitly mentioned evolution as a topic he missed, i.e., *'how to maintain and evolve a product line in practice'*. Three full-time students, however, also commented that they would have wanted more practical content. For instance, one answered that he would have wished for *'more case studies and more live examples taking a very complex product and analyzing it and then taking it through the whole course and developing the product line'*.

While almost half of all respondents said they learned about case studies, the majority wanted to learn even more about practice. One student said that he *'would have wished for more links between model-driven engineering and SPL engineering'*. Along the same lines, other students mentioned that they would like to see covered basic relationships with more recent topics such as service orientation or cloud computing, while others added that they missed the economics and business side of SPLs. One student also commented that *'SPL engineering often seems to be presented as a (too) ideal approach to develop software but does not (enough) take into account the reality that existing (legacy) software still needs to be maintained and/or refactored into an SPL'*.

4.2.6. How students define SPL after the course. Asking students to define in their own words what a Software Product Line is led to some interesting replies. Many students (10) define SPL engineering as an approach to systematic reuse, for example:

- *'it is a means to produce reusable assets systematically and to increase reusability of features shared by a family of products'*.
- *'SPLs are initiatives to provide software reuse for a domain in a systematic way'*.
- *'a software engineering paradigm used to develop a set of products within a given domain. It makes use of commonality and variability aspects of such products to enable systematic reuse of artifacts'*.

Four respondents focused their definition more on the aim of SPL engineering to quickly generate products, e.g., *'A SPL encompasses features [...] with the goal to generate different products from the SPL'*. Three students centered their answers around the goal of minimizing development effort and cost, e.g., *'SPL is company strategy for the production of [...] software products that share some similarity to minimize the cost and effort of development'*. Three students focused on the configuration or composition aspects of SPLs. One student raised the issue that a SPL does not only involve code but also documentation, another drew a similarity to the term product line in industrial manufacturing. Another student from industry provided a revealing comment: *'An interesting but immature approach to handle different variations of software products'*.

4.2.7. Perceived challenges of learning about SPLs. Students see several challenges regarding learning about SPLs (which interestingly can be mapped with the challenges identified by scholars quite well). Seven students (all without industry job) think that *'it is very challenging to understand the general concept of (large-scale) reuse as a student'*. Three others (one with and two without industry job) commented that understanding the basic concepts of SPL engineering, especially *'imagining how it will work in practice is difficult'*. This again points to the need for more case studies and practical examples suitable for teaching SPLs. Four students (two with an industry job

and two full-time students) replied that a lot of background knowledge is required to understand SPLs, especially a lot of experience in software engineering. This is also confirmed by the teaching survey. Again, case studies were mentioned here as essential (by two of these four students). Two full-time students had *difficulties understanding 'cross-cutting concerns' and 'feature interactions'*.

Other challenges mentioned by one student each were:

- the *visualization* of variability.
- understanding the *mapping of abstract features to concrete software artifacts* (i.e., impacts of selecting a feature).
- *release and support of variants*.
- *automation* of feature modeling and product configuration.
- *maintenance* of the many variants created (derived) based on a product line.
- the available *tooling*.

4.2.8. Suggestions to improve teaching SPLs. Students had the following suggestions to improve the state of the teaching of SPLs (11 students had no suggestions or said that nothing needs to be improved). Six students (five with industry jobs and one full-time student) suggested to *'include more practice activities and discussions and details on how to transfer the theoretical knowledge to the real world'*. One even suggested *'agreements with companies'* should be made to improve teaching SPLs by directly involving companies, similar to replies we received in our teaching survey. One industry student suggested to *'let students try out commercial SPL tools'*. One full-time student suggested to *'also explicitly focus on SPL development in small organizations, not only large industrial companies'*. Another suggestion was to *'compare SPL engineering with other recent approaches such as agile development'*. Three students suggested creating *'more open sources for SPL teaching such as a MOOC (Massive Open Online Course)'*, while the other three also suggested *'providing detailed solutions to practical exercises'* that can help students compare and contrast their understanding.

4.2.9. Impacts on industrial practice. We also asked students how the teaching of SPLs impacts industrial practice. Here it is particularly important to distinguish answers by students who already work in industry and full-time students and also to take into account the countries where students participated in the courses (because of a potentially different industrial landscape).

Again, five students (all with an industry job) brought up case studies as a key argument, i.e., *'knowledge about concrete case studies and success stories'* can be used by students in industry to convince their managers about the usefulness of SPL engineering. The other replies were quite diverse. Two students (both from industry) commented that *learning about SPLs in general already motivated them to think about using SPL techniques for new projects in their job*. Another industry student said it helped him a lot during interviews and that it *broadened his view on software development in general*.

One full-time student commented that s/he assumes *'learning about SPLs could reduce the amount of training needed when hiring developers'*. Some other comments centered around the fact that, with a paradigm shift in industry from single systems engineering to SPL engineering, the learning about SPLs is crucial for students. Two students pointed out that *the teaching of SPLs might also benefit tool vendors*, and that adequate tool support can ease adoption of SPLs in industry.

4.2.10. Other comments. Further comments by students were that a survey on learning about SPLs is an excellent initiative to improve SPL teaching and technology transfer. This is an important point to be considered, i.e., instead of general evaluation forms handed out to students typically (not focusing much on the concrete topic of the course), *the state of teaching should be surveyed more often, also in other areas*.

Another comment by a student was that *'it would be a good idea to work on one big, concrete project throughout the semester to experience the development of a SPL from scratch'*. Three students (one with an industry job and two full-time students) again pointed out the necessity to include more case studies. Other students expressed their preferences for concrete SPL approaches such as

feature-oriented SPL development or aspect-oriented SPL development and particular tools (cf. the tools table of our teaching survey).

4.3. Threats to Validity

A first threat to validity is that the survey was designed by just three researchers, based on their experience in research and teaching SPLs. To mitigate this threat, we asked colleagues for feedback and refined the surveys to incorporate their feedback before sending them out. We also test-drove it with students in our own courses to refine questions and possible answers.

Another threat regards the low number of responses we received. For the student survey 122 invitations were made and only 25 students provided complete responses and 37 provided partial responses. Obviously, we could not properly motivate students to reply. The rather low number of (complete) responses (25) decreases the usefulness of any quantitative analysis. We thus decided to focus more on analyzing qualitative results in which we also took partial responses into account.

A further threat to the internal validity of our results is that participants misunderstood questions. We mitigated this threat by test-driving the questionnaires with students during their development. Also, the open questions at the end of the surveys allowed participants to raise concerns and ask for clarification. Another (big) threat is that some participants might not have had the required knowledge to answer all of our survey questions. For example, we asked students what else they wanted to learn and what were their expectations before taking the course; such questions assume knowledge students may not have. Again, we addressed this threat through the possibility of adding comments to the survey, e.g., don't know, and only counted complete responses.

An external threat is that, perhaps, we did not contact a representative pool of students. In addition to selecting students from our own courses, we also re-contacted the participants of our teaching survey and asked them to send the survey about learning SPLs to their students. As we wanted the student surveys to be anonymous, we can not report the distribution of students from the second survey among the scholars from the first survey. However, the results of the learning survey indeed have a strong focus on the situation in European countries and Brazil. For instance, answers on the question about the impact of teaching SPLs on industrial practices depend a lot on the industry in a particular country. We tried to take this fact into account in our interpretations of open questions. Also, we compared responses to the survey by students from different locations to check whether the answers are too divergent (they are not).

5. SPLTEA WORKSHOPS

As a follow up of the two surveys involving scholars and students, we organized additional workshops to collect additional feedbacks from the SPL community.

5.1. Research Method

Having received very positive feedback from the community for our survey on SPL teaching [Acher et al. 2014a], we decided to organize a first workshop at the 18th International *Software Product Line Conference (SPLC)* in 2014 [Acher et al. 2014b]. The goal was to further explore the current status and ongoing work on teaching SPLs and variability at universities, colleges, and in industry. Via an open call for papers, we invited submissions reporting research on SPL teaching, discussing experiences with SPL teaching, as well as position and vision papers (see http://spltea.irisa.fr/2014/). Three experience report papers were eventually accepted and presented at the workshop. About 30 people participated in the workshop and many interesting discussions took place, especially during an industry panel we organized as part of the workshop program inviting four experienced industry practitioners to discuss their views of SPL teaching.

At SPLC 2015, we again organized the workshop (http://spltea.irisa.fr/) to further discuss important issues [Acher et al. 2015]. This time we did not focus on soliciting papers but put a special focus on the design and population of an open repository of resources dedicated to SPL teaching (http://teaching.variability.io). The need to create such a repository had been identified in the sur-

veys as well as at the first workshop and we dedicated the second workshop to address this need. About 20 people participated in the workshop.

Below we summarize the experience reports published at SPLTea 2014, the industry panel held at SPLTea 2014, as well as the discussions from SPLTea 2014 and 2015 as a further input to our discussion of challenges, perspectives and concrete recommendations that we will summarize in the next section. Please note that our summary of experience reports, the industry panel, as well as discussions that took place at the two workshops obviously only can provide further qualitative input to our results. However, we think it helps to further validate what we found out in our surveys.

5.2. Experience Reports (SPLTea 2014)

Seidl and Domachowska presented their experiences on 'Teaching Variability Engineering to Cognitive Psychologists' [Seidl and Domachowska 2014]. They describe a curriculum to teach variability engineering techniques (such as feature modeling) to cognitive psychologists to enable them to cope with variability inherent in the protocols to design and perform their experiments. Their first experiences show that variability engineering can be successfully used by psychologists; however, because of their different educational background, topics that required more programming or mathematical skills proved more challenging for them to grasp.

McGregor reported about 'Ten Years of the Arcade Game Maker Pedagogical Product Line' [McGregor 2014]. McGregor emphasizes that teaching about SPLs covers both software engineering practices and management practices, which is challenging for students as they are either engineering-focused or management-focused, but not both. Nevertheless, 'a student cannot adequately understand the software product line strategy without an exposure to a breadth of engineering and management practices'.

Collet et al. presented their 'Experiences in Teaching Variability Modeling and Model-driven Generative Techniques' [Collet et al. 2014]. They observed that it is indeed very difficult to introduce SPL engineering techniques in a software engineering course. In addition to the known issue of lack of case studies and robust tools, they found that the major challenges for SPL teaching are the inherent complexity of the subject and the strong background on software system development needed to fully appreciate the benefits of SPL engineering. Hence, their work corroborates the findings of our teaching survey [Acher et al. 2014a].

Subsequently the same authors presented an experience paper at the Educator's Symposium (MODELS'14) [Mosser et al. 2014]. This time, the paper focuses on issues related to the teaching of modeling (in the same context as in [Collet et al. 2014]). They emphasize the need of a non-trivial and realistic case study for better motivating students and illustrating MDE concepts. A similar need has been reported in our teaching survey by half of the participants.

5.3. Industry Panel (SPLTea 2014)

The SPLTea 2014 workshop also included a panel on SPL teaching and industry needs. This panel included: Dr. Paul Clements, VP of customer success at BigLever (a leading commercial SPL engineering tools provider); Dr. Martin Becker, head of Embedded Systems Development at Fraunhofer IESE; Dr. Steve Livengood, Software Architect at Samsung Research America; and Dr. Juha Savolainen, SW Director at Danfoss Power Electronics. These four panelists are leading practitioners of SPLs in industry, whose extensive and broad experience dealing with educating engineers and managers in industrial settings puts them in a unique position to appreciate the challenges and needs of industry regarding SPL teaching. Below we summarize the key points discussed by the panelists and the audience, revolving around the following issues.

How should SPLs be taught so that students going to industry have the required knowledge? First and foremost they should have a strong and proficient background on software development. Once they are armed with that skill foundation the transition to SPLs becomes easier. The first hurdle is understanding why SPLs are needed and what technologies are available to approach them. Product line engineering infuses additional complexity in the software engineering problems

– and students should be aware of it. Sometimes practitioners '*have the impression to already have done the job but they do not put a name on it*'.

What is the required SPL knowledge for industry, e.g., how relevant are variability models? A first challenge that must be tackled is the use of a common and coherent terminology that is understandable for all the stakeholders involved. The cornerstone concepts that must be taught are *core and shared assets* as everything else revolves around these concepts. Careful attention should be paid to avoid using ambiguous or imprecise concepts. The term *version* was given as an example. It might have different meanings: a temporal dimension (variability in time, as in releases or versioning systems) or a variant dimension (variability in space). The expectation from an engineer working in a product line context is to '*treat variability in the same way as other basic functional and non-functional properties are treated.*'

What are the challenges of teaching SPLs in the light of the needs of the industry? SPLs add another layer of complexity to software development. However, the engineers with the right background and mindset can easily understand SPL ideas and adopt them to their practices. Quite often the most challenging part is convincing the management hierarchies that are far from the development front. One phrase to summarize the situation was put forward by a workshop participant '*technology is easy, people are not*'. Also, the challenge of '*keeping alive*' product lines is hard to teach. The decision whether an SPL has to be maintained or has to evolve impacts both economical, organizational, and technological aspects of a company.

How can the state of teaching SPLs be improved? In short, making the courses more 'realistic'. In other words, so that the size and complexity of student projects closely reflects industrial practices. Teaching product lines '*as a whole*' is highly challenging: it is difficult to find a comprehensive, end-to-end project. From this perspective, the panel argues that more pedagogical case studies like the 'Arcade Game Maker Pedagogical Product Line' [McGregor 2014] are missing. The ideal scenario would be a direct involvement of companies providing actual cases for development throughout courses. Unfortunately, issues such as intellectual property get in the way for this scenario to happen. Nonetheless, any mechanisms that can help promote the involvement of industrial companies should indeed be thoughtfully considered. An interesting report from the audience was the organization of a *workshop* involving a company developing SPLs and students (in addition to teachers and researchers).

Another issue raised was elevating variability modeling and management as core concepts in standard software engineering curricula, at the same level of properties such as reliability, performance, or scalability that are deemed crucial and standard for software development. Achieving this goal would require a concerted effort of the SPL community and the software engineering research communities at large. Concrete actions would be advocating the inclusion of variability management issues in regular software engineering courses and teaching materials.

How could industry be better involved in teaching SPLs? Industry involvement in the teaching of SPLs should be seen as a mutually beneficial activity. Industry gets help addressing technical and managerial challenges, while academia prepares students to address such challenges. For such an interaction to happen, there needs to be a strong and vocal advocate from the industry side that not only convinces managers of the benefit of the interaction with academia but also provides guidance and mentoring to students so they can recognize the needs of the industry faster and better, and consequently start yielding results sooner, i.e., a champion is needed [Wohlin et al. 2012].

Also, product line tools used in industry can be used within the context of education. Companies represented in the panel are open to share their tools with academic organizations. Both, BigLever and pure::systems offer academic/educational licenses (on request).

Other discussion points. The panelists expressed their perception that the culture of SPL adoption in industry has gradually yet dramatically changed from its beginning around a couple of decades ago. In the past, companies were unwilling to publicly acknowledge their use of SPL technologies because of their concern of loosing competitive advantage in their market, as their competitors might also try to use SPLs for their benefit. Now, the attitude has shifted to the point that they instead not only publicly acknowledge their use but go as far as to advertise it. In other words, there is a trend of

perceiving SPL practices as standard and mainstream. This is the best scenario that the SPL research and practitioners communities can hope for. Teaching and education can be two catalysts for this scenario to materialize.

5.4. Workshop Discussions (SPLTea 2014 and 2015)

Beyond the experience reports presented at the first workshop, the participants of the 2014 and the 2015 workshops discussed the following major topics.

Need for case studies. First and foremost the lack of case studies was a major gap raised in the discussion. Again this identified need corroborates the findings of our first [Acher et al. 2014a] and second survey. It was agreed during the workshops that the main characteristic of a good case study is that it should be realistic, meaning of enough size and complexity that helps students understand what are the problems faced, why SPLs are needed, and what technological and methodological solutions are currently available. Some of the participants argued that the case study should cover each phase of the software development lifecycle – as variability impacts all phases.

The Arcade Game Maker Pedagogical Product Line [McGregor 2014] was originally designed with this purpose in mind. Suggestions were made about looking at Open Source projects as potentially ideal candidates for case studies for teaching. However, this type of systems might not be originally conceived as SPLs (hence might not represent the real SPL engineering in practice in industry) or their development process might not be fully accessible or well documented to be understandable by people outside their respective developers and contributors.

Another suggestion to address this gap was the coordination of a multi-site software development project, similar to those carried out in the Global Software Engineering community (also see [Beecham et al. 2017]), where groups of students from several universities develop a joint project. For instance, the participating groups could develop different variants of a system that could subsequently be the foundation to build a SPL. Another option is to develop inside the curriculum a common and large case study combining different disciplines of software engineering as reported by [Collet et al. 2014; Mosser et al. 2014].

Curriculum and different ways to teach SPL knowledge. There was a discussion on how to better integrate SPL teaching into the standard computer science curriculum. An ideal alternative is creating a dedicated course for the subject. This might be feasible at the graduate level where the courses are expected to be more specialized. However, at the undergraduate level, there is a strong competition with other subjects that might be deemed more relevant or marketable such as cloud computing, big data, or visualization.

A second alternative is the inclusion into a software engineering course, such as software design or software development, that are typically part of any undergraduate study in the area. The downside of this second alternative is that the amount of time and topics that can be devoted to SPL is reduced and consequently the understanding and appreciation of SPL engineering by the students is negatively affected.

A third alternative suggested was to spread SPL topics across specialized courses, i.e., addressing SPL testing issues within an specialized software testing course where a few sessions and practical exercises are devoted to SPL. Though attractive, this third alternative requires good knowledge of SPLs for all the instructors involved and also a very good curricular coordination that might be difficult to attain in institutions with flexible curriculum or large student populations.

Populating a repository of teaching material. At SPLTea 2015 (second edition of the workshop), we solicited participants to discuss about a repository of teaching material for SPLs. There were about 20 participants forming two distinct groups of SPL educators, researchers, and practitioners. Each group discussed various questions like: What could be the desirable features, services, and underlying technology of such a repository? The main ideas and insights can be summarized as follows.

First, there is a need for educators to reuse and adapt *existing* material (slides, exercises, case studies, syllabus, etc.). Starting a new course from scratch is a difficult and time-consuming activity. On the other hand most participants argue that the content of the course depends on several factors

(place in the curriculum, prior knowledge of students, etc.): The customization of existing resources is clearly needed and a basic use case is to pick and assemble some material to help constituting a complete course. Participants coming from industry also need to educate and train engineers. Similarly they need to customize material to their particular audience and project.

Second, there is no consensus on the *scope* of the repository. Some participants have a focus on variability modeling and software implementation issues. Others tend to have a broader perspective and include business organizational aspects of SPL engineering. Overall the repository acts as an interesting opportunity to refine or complement existing *curriculum* (see previous point).

Third, participants believe the repository can bring to the foreground a comprehensive case study for illustrating SPL concepts. The reuse of case studies, based on existing material, is a possible direction. Educators could improve their current proposals based on other experiences. The repository is seen as a central network for sharing educators' knowledge and students' feedbacks.

Fourth, an interesting idea is to involve students in the process of populating the repository. That is, students could correct, extend, or create new material of the repository. It can help to improve the quality of the material and engage students when learning SPLs.

Finally, we discussed technical aspects of the repository as well as licensing issues.

After SPLTea'15, we set up a solution based on Github features (e.g., pull requests): http://teaching.variability.io. We could already collect some SPL teaching resources in the repository. The further population as well as the ability of the repository to help in some teaching issues (e.g., curriculum, case studies) are work in progress.

5.5. Threats to Validity

An external threat to the validity of our discussions of the workshops is that, perhaps, we did not have a representative set of participants in our teaching workshops. We do claim representativeness of workshop participants as over 30 participants from the SPLC community, many of them senior researchers and industry people, participated in the discussions in the workshops. For the industry panel held at the first workshop, we carefully selected industry panelists with many years of experience in SPL engineering and teaching to further mitigate the threat regarding representativeness.

Our possibly wrong interpretation of discussions is a further important threat. We are confident, however, that we could capture the essence of the discussions in our interpretations. We could back up most of discussions at the workshops with the answers in the surveys. Also, we compared responses to the teaching survey by industry people with responses by academics to check whether the answers are too divergent (they are not).

While participation at the workshops was good, we could not attract many papers (only 3 at the first workshop). Participation shows a deep interest in the topic. Yet, the low number of papers shows that it is hard to come up with a paper, i.e., do research about it. It calls for more incentives (see next section).

6. RECOMMENDATIONS FOR EDUCATORS

Building upon the previous empirical results (from the two surveys on teaching and learning SPLs and the two workshops), we put forward practical advices and recommendations for SPL educators as well as for the SPL community.

SPL tools are here. Without tools, educators can have severe difficulties for illustrating concepts, practicing or evaluating students, running a project, or connecting an SPL course to others – up to point they give up the idea of building an SPL course. While this issue has been raised in the teaching survey, it has not really been perceived as an issue in the learning survey, only one comment on (un)availability of commercial tools was made. Also, in the workshops, this was not seen as a problem, due the possibility to get academic licenses for both major commercial SPL tools (pure::variants and Gears). Furthermore, (open source) research prototypes such as FeatureIDE [Thüm et al. 2014], FAMILIAR [Acher et al. 2013], BUT4Reuse [Martinez et al. 2015], or Clafer [Bak et al. 2016] become increasingly useful. Material we collected so far in our teaching

repository, the existing SPL text books as well as experience reports presented at SPLTea'14 used (some of) these tools.

Key message: Tools are no longer a source of anxiety and an obstacle: educators now have a collection of tools for modeling and implementing SPLs at their disposal.

Curriculum integration: many possible scenarios. Teaching SPLs alone has no sense. There are pre-requisites needed and SPL engineering should be connected to other courses. That raises the question of how to integrate SPL into a curriculum. This was an issue raised in the teaching survey and the workshops. It was not raised in the learning survey, but the answers of students show that some of them did indeed not open their minds, i.e., that it was hard for them to grasp the basic concepts of systematic reuse. Our empirical investigations show that there are at least three possible scenarios for integrating SPL into a curriculum:

(1) Dedicated SPL course. First, an alternative we consider ideal is creating a dedicated course for the subject. However, at the undergraduate level, there is a strong competition with other subjects that might be deemed more fundamental, relevant or marketable such as cloud computing, big data, or visualization. Creating a dedicated course is thus more feasible at the graduate level where the courses are typically more specialized. We thus recommend to design an SPL course at a graduate level, also since many skills are needed to apprehend SPL concepts. Another recommendation is not to start the construction of an SPL course from scratch but rather reuse existing material. Beyond the text books, instructors can find examples of comprehensive SPL courses in our teaching repository. Moreover, as suggested at SPLTea'15, instructors can build their own courses by picking and composing materials, depending on the emphasis (e.g., variability modeling, variability implementation, or the whole SPL process) and pre-requisites of the course.

(2) Include in course at the undergraduate level. A second alternative that was discussed is the inclusion into a software engineering course, such as software design or software development, that is typically part of any undergraduate study in the area. Specific SPL concepts or techniques can then be presented. Feature modeling can be used as part of an introduction to software modeling, in complement to the traditional use of UML diagrams. When presenting design patterns, the emphasis can be made on customization, reuse, and variability. The downside of this second alternative is that the amount of time and topics that can be devoted to SPLs is reduced. Therefore educators should have a rather modest objective in terms of SPL understanding by undergraduate students. It should be seen as a first step towards more investigations at the graduate level.

(3) Embedding SPLs. A third option suggested was to spread SPL topics across specialized courses, for instance, addressing SPL testing issues within an specialized software testing course where a few sessions and practical exercises are devoted to SPL. This third alternative requires good knowledge of SPLs for all the instructors involved, which can be a problem. However, in practice, SPL educators are usually experts and involved in other areas (e.g., model-driven engineering or requirements engineering). A main advantage for instructors is to show the applicability of SPL techniques and their relations to other software aspects. Another positive aspect of embedding SPLs into an existing course is that more integrated and ambitious projects can be considered (see hereafter).

Key message: The design of a full SPL course, though feasible and already often done, is not the only option for educators. Another realistic alternative is to teach about SPLs as part of existing courses. Experience reports as well as discussions at the SPLTea workshops suggest to teach SPLs at the graduate levels; attempts at the undergraduate levels are more exploratory at the current state of practice.

From exercises to case studies. SPL engineering involves numerous activities, including requirements analysis, design, implementation, testing, modeling, and evolution. Students should understand the underlying problems and solutions in practical sessions. Educators have at their disposal a collection of resources to teach individual topics in books or online. For instance, exercises for feature modeling or reverse engineering SPLs are available in our teaching repository. Though well-focused exercises are useful, more substantial case studies can be considered for apprehending the whole SPL engineering process.

In fact, the lack of and availability of well-documented real-world case studies suitable for teaching was a key issue raised by scholars, students, researchers, and industry practitioners. We have discussed this issue throughout this article at length already. Though there are difficulties, educators can still use existing case studies from text books and research papers. Our first recommendation is to only consider a subset of SPL activities (e.g., variability modeling and implementation) since treating all SPL activities at once is certainly a too ambitious objective. A second recommendation is to connect an SPL project to other software aspects (e.g., testing), which leads to more substantial, realistic, and engaging cases. A recent example of a project combining Web development, model-driven engineering, and SPL engineering is reported in [Halin et al. 2017].

Key message: Specific exercises can be reused for teaching individual SPL topics such as scoping, feature modeling, variability implementation, SPL validation and verification, etc. Despite early efforts, encompassing all SPL activities within an unique case study is an open and difficult challenge. Therefore it is more realistic to only consider a subset of SPL activities. Another trend is to develop a running SPL project in coordination with other courses, which leads to more substantial, realistic, and engaging cases.

A baseline curriculum and evaluation scheme. It seems unrealistic to create one standard curriculum for teaching SPLs worldwide. For instance, no respondent reported on a full curriculum based on SPL engineering. Yet, having a baseline for creating curricula and performing student evaluations would still be helpful for the community. An inspiring example is a curriculum proposal on Data Mining put forward by the respective research community [Chakrabarti et al. 2006]. For an SPL course, based on our personal experiences and by analyzing SPL books and courses online, we can suggest to center a SPL course around the following topics (cf. the material collected in our repository at http://teaching.variability.io so far):

- variability management and (feature) modeling.
- variability implementation and feature-oriented software development.
- product line (domain) definition/scoping.
- product line (domain) design.
- product line (domain) testing.
- product derivation.
- tools for product line engineering.
- product line evolution and reverse engineering product lines.
- product line case studies.

Exercises should include:

- creating feature/product maps (product line scoping).
- developing (different types of) variability models.
- implementing features (feature-oriented software development).
- assessing and discussing existing product line case studies.

Key message: Together with tools, exercises, cases, and reports, educators can build upon our baseline for designing their SPL courses. Depending on the educational context and curriculum integration (see previous points), we recommend to either consider some SPL topics as optional or to specifically investigate them within an existing course.

Growing the body of knowledge in SPL teaching. We have made a series of concrete recommendations, based on the *current* state of practice. However, the way SPLs are taught can still be improved and many limitations have been reported by educators, students, or industry participants.

Hence, an additional advice we have is that SPL educators should share their exercises, tools, case studies, positive or negative experiences. In general they should communicate with other SPL educators and participate within a community. We have made preliminary steps with the SPLTea workshops and with the development of a virtual meeting place: http://teaching.variability.io. We naturally invite educators to contribute in this repository. We hope to gather more SPL material but also to improve existing ones.

Beyond our initiative, the SPL community should encourage contributions in the area of SPL education. The SPLC'17 conference explicitly calls for papers for the topic "Education and SPL" in the main tracks. Other initiatives, such as tool and artifact recognition, can also be useful for improving SPL teaching material.

Key message: Overall, the body of knowledge in SPL education should grow. Educators have a key role to play through the discussion, sharing, and dissemination of their experiences and materials.

Challenges ahead. Teaching of SPLs is still in its infancy. It is no surprise numerous open questions exist. For instance, it is unclear whether and how variability should be elevated as core concept in a standard software engineering curriculum; we have not yet found a recipe for successfully integrating and engaging industry in our courses; we do not have yet a common case study. Though some ideas and discussions have arisen, there is no clear answer. From several perspectives, the current SPL teaching practices are not mature enough – and the same observation has been made for other computer science disciplines (e.g., [Paige et al. 2014; Whittle and Hutchinson 2011; Bagge et al. 2014; Cooper and Cunningham 2010]). It is more than time to collect more experiences and evidence.

Key message: Our general recommendation is that SPL educators should innovate, take risks, experiment, possibly deviate from current practices, and eventually report on their failures or successes to the community.

7. CONCLUSIONS AND FUTURE WORK

We presented the results of three initiatives we conducted to assess the state of practice of teaching software product lines (SPLs). The resulting snapshot is based on evidence of four kinds of actors involved in education and SPLs:

- **educators:** 35 responses of an online survey allowed us to report quantitative as well as qualitative results about the context of teaching SPLs (including length of courses, primary literature used, tools used, and parts of the SPL development lifecycle covered) as well as about the experience of SPL teaching (including challenges of teaching SPLs and suggestions to improve the state of teaching SPLs). Moreover, we summarized discussions related to three experience reports presented at SPLTea 2014, a workshop we organized at the 2014 edition of the Software Product Line Conference, and SPLTea 2015, a workshop we organized at the 2015 edition of the Software Product Line Conference.
- **researchers:** 30 of 35 responses of the previous online survey were related to the impacts of teaching on SPL research.
- **students:** 25 responses of another survey allowed us to find out about students' experiences with learning about SPLs, providing information about what and how they learned about SPLs.
- **industry practitioners:** We summarized the discussions of a panel held at the SPLTea 2014 workshop focusing on SPL teaching and industry needs.

We discussed how to continue improving teaching SPLs, i.e., creating a virtual meeting place for the community (e.g., http://teaching.variability.io/) and developing a baseline curriculum. Building upon empirical results, we made concrete recommendations for SPL educators. On the one hand, the numerous tools, textbooks, or exercises available here and there are an interesting starting point for building SPL courses and integrating them into a curriculum. On the other hand, we encourage SPL educators to share their experiences and innovate with the long term goal of improving current SPL practices.

The diversity of skills, tools, techniques, and paradigms students have to learn continue to grow. We hope future actions will help to better understand how SPLs and other specialized fields in computer science can be integrated into existing curricula.

ACKNOWLEDGMENTS

First and foremost we would like to thank the participants of the surveys and the workshops as well as the colleagues that provided us with valuable feedback on earlier drafts. We thank the anonymous reviewers for their careful reading and insightful suggestions. Lopez-Herrejon's work was partially funded by the Austrian Science Fund (FWF) Lise Meitner Fellowship M1421-N15. Rabiser's work is supported by Primetals Technologies and the Christian Doppler Forschungsgesellschaft, Austria.

REFERENCES

Ebrahim Khalil Abbasi, Arnaud Hubaux, Mathieu Acher, Quentin Boucher, and Patrick Heymans. 2013. The Anatomy of a Sales Configurator: An Empirical Study of 111 Cases. In *Proc. of the International Conference on Advanced Information Systems Engineering (CAiSE'13)*. Springer, Valencia, Spain, 162–177.

Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2013. FAMILIAR: A domain-specific language for large scale management of feature models. *Sci. Comput. Program.* 78, 6 (2013), 657–681.

Mathieu Acher, Roberto Lopez-Herrejon, and Rick Rabiser. 2014a. A Survey on Teaching of Software Product Lines. In *Proc. of the 8th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2014)*. ACM, Nice, France, 3–10.

Mathieu Acher, Roberto E. Lopez-Herrejon, and Rick Rabiser. 2014b. SPLTea 2014: First International Workshop on Software Product Line Teaching. In *18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15-19, 2014*. 352.

Mathieu Acher, Roberto E. Lopez-Herrejon, and Rick Rabiser. 2015. SPLTea 2015: Second International Workshop on Software Product Line Teaching. In *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*. 396.

Vander Alves, Nan Niu, Carina Alves, and George Valenca. 2010. Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology* 52, 8 (2010), 806–820.

Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*. Springer.

Anya Helene Bagge, Ralf Lämmel, and Vadim Zaytsev. 2014. Reflections on Courses for Software Language Engineering. In *Proc. of the 10th Educators' Symposium @ MODELS 2014 (EduSymp'14), Workshop*. Springer LNCS, Valencia, Spain, 1–10.

Kacper Bak, Zinovy Diskin, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. 2016. Clafer: unifying class and feature modeling. *Software and System Modeling* 15, 3 (2016), 811–845. DOI:http://dx.doi.org/10.1007/s10270-014-0441-1

Sarah Beecham, Tony Clear, John Barr, Mats Daniels, Michael Oudshoorn, and John Noll. 2017. Preparing Tomorrow's Software Engineers for Work in a Global Environment. *IEEE Software* 34, 1 (2017), 9–12.

David Benavides, Sergio Segura, and Antonio Ruiz Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636.

Nelly Bencomo, Svein O. Hallsteinsen, and Eduardo Santana de Almeida. 2012. A View of the Dynamic Software Product Line Landscape. *IEEE Computer* 45, 10 (2012), 36–41.

Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013a. A Survey of Variability Modeling in Industrial Practice. In *Proc. of the Seventh International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2013)*. ACM, Pisa, Italy, 7–14.

Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. 2013b. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Transactions on Software Engineering* 39, 12 (2013), 1611–1640.

Jan Bosch and Petra Bosch-Sijtsema. 2010. From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software* 83, 1 (2010), 67–76.

Soumen Chakrabarti, Martin Ester, Usama Fayyad, Johannes Gehrke, Jiawei Han, Shinichi Morishita, Gregory Piatetsky-Shapiro, and Wei Wang. 2006. Data Mining Curriculum: A Proposal (Version 1.0). (April 2006). http://www.sigkdd.org/curriculum/index.html

Lianping Chen and Muhammad Ali Babar. 2011. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology* 53, 4 (2011), 344–362.

Paul Clements and Linda Northrop. 2001. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering, Addison-Wesley.

Philippe Collet, Sébastien Mosser, Simon Urli, Mireille Blay-Fornarino, and Philippe Lahire. 2014. Experiences in teaching variability modeling and model-driven generative techniques. In *Proc. of the 18th International Software Product Line Conference (ICSE 2014): Companion Volume for Workshops, Demonstrations and Tools-Volume 2*. ACM, Florence, Italy, 26–29.

Steve Cooper and Steve Cunningham. 2010. Teaching Computer Science in Context. *ACM Inroads* 1, 1 (March 2010), 5–8.

CVL. 2014. Common Variability Language (CVL). http://www.omgwiki.org/variability/doku.php?id=start. (2014).

Krzysztof Czarnecki and Ulrich W Eisenecker. 2000. *Generative Programming: Methods, Techniques, and Applications*. Addison-Wesley.

Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wasowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proc. of the 6th International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS 2012)*. ACM, Leipzig, Germany, 173–182.

Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D. McGregor, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. 2011. A systematic mapping study of software product lines testing. *Information and Software Technology* 53, 5 (2011), 407–423.

Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *Proc. of the 17th European Conference on Software Maintenance and Reengineering (CSMR 2013)*. IEEE, Genova, Italy, 25–34.

Emelie Engström and Per Runeson. 2011. Software product line testing – A systematic mapping study. *Information and Software Technology* 53, 1 (2011), 2–13.

Carlo Ghezzi and Dino Mandrioli. 2005. The challenges of software engineering education. In *Proc. of the International Conference on Software Engineering (ICSE)*. Springer, St. Louis, MO, USA, 115–127.

Hassan Gomaa. 2005. *Designing Software Product Lines with UML*. Addison-Wesley.

Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Patrick Heymans. 2017. Yo Variability! JHipster: A Playground for Web-Apps Analyses. In *Proc. of the 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2017)*. ACM, Eindhoven, The Netherlands, 44–51.

Mike Hinchey, Sooyong Park, and Klaus Schmid. 2012. Building Dynamic Software Product Lines. *IEEE Computer* 45, 10 (2012), 22–26.

Gerald Holl, Paul Grünbacher, and Rick Rabiser. 2012. A Systematic Review and an Expert Survey on Capabilities Supporting Multi Product Lines. *Information and Software Technology* 54, 8 (2012), 828–852.

Arnaud Hubaux, Andreas Classen, Marcílio Mendonça, and Patrick Heymans. 2010. A Preliminary Review on the Application of Feature Diagrams in Practice. In *Proc. of the 4th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10)*. ICB Research Report, Universität Duisburg-Essen, Linz, Austria, 53–59.

Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2011. A Survey of Empirics of Strategies for Software Product Line Testing. In *Proc. of the IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST) Workshops*. IEEE, Berlin, Germany, 266–269.

Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Technical Report CMU/SEI-90TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

Ludwik Kuzniarz and Luiz Eduardo G Martins. 2016. Teaching Model-Driven Software Development: A Pilot Study. In *Proceedings of the 2016 ITiCSE Working Group Reports*. ACM, 45–56.

Daniela Lettner, Michael Petruzelka, Rick Rabiser, Florian Angerer, Herbert Prähofer, and Paul Grünbacher. 2013. Custom-Developed vs. Model-based Configuration Tools: Experiences from an Industrial Automation Ecosystem. In *Proc. of MAPLE/SCALE 2013, Workshop at the 17th International Software Product Line Conference (SPLC 2013)*. ACM, Tokyo, Japan, 52–58.

Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. 2010. An analysis of the variability in forty preprocessor-based software product lines. In *Proc. of the 32nd International Conference on Software Engineering (ICSE 2010)*. IEEE, Cape Town, South Africa, 105–114.

Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Bottom-up adoption of software product lines: a generic and extensible approach. In *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*. ACM, 101–110.

John D. McGregor. 2014. Ten years of the arcade game maker pedagogical product line. In *Proc. of the 18th International Software Product Lines Conference (SPLC 2014) - Companion Volume for Workshop, Tools and Demo papers*. ACM, Florence, Italy, 24–25.

Brice Morin, Olivier Barais, Jean-Marc Jézéquel, Franck Fleurey, and Arnor Solberg. 2009. Models@ Run.time to Support Dynamic Adaptation. *IEEE Computer* 42, 10 (2009), 44–51.

Sébastien Mosser, Philippe Collet, and Mireille Blay-Fornarino. 2014. Exploiting the Internet of Things to Teach Domain-Specific Languages and Modeling. In *Proc. of the 10th Educators' Symposium @ MODELS 2014 (EduSymp'14), Workshop*. Springer, Valencia, Spain, 1–10.

Richard F. Paige, Fiona A. C. Polack, Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, and James R. Williams. 2014. Bad Modelling Teaching Practices. In *Proc. of the 10th Educators' Symposium @ MODELS 2014 (EduSymp'14), Workshop*. Springer, Valencia, Spain, 1–12.

Klaus Pohl, Günter Böckle, and Frank van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer.

Rick Rabiser, Paul Grünbacher, and Deepak Dhungana. 2010. Requirements for Product Derivation Support: Results from a Systematic Literature Review and an Expert Survey. *Information and Software Technology* 52, 3 (2010), 324–346.

Christoph Seidl and Irena Domachowska. 2014. Teaching variability engineering to cognitive psychologists. In *Proc. of the 18th International Software Product Lines Conference (SPLC 2014) - Companion Volume for Workshop, Tools and Demo papers*. ACM, Florence, Italy, 16–23.

Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Sci. Comput. Program.* 79 (2014), 70–85.

Frank van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer Berlin Heidelberg.

Michael Vierhauser, Rick Rabiser, and Paul Grünbacher. 2014. A Case Study on Testing, Commissioning, and Operation of Very-Large-Scale Software Systems. In *Proc. of the 36th International Conference on Software Engineering (ICSE), ICSE Companion*. ACM, Hyderabad, India, 125–134.

David Weiss and Chi Tau Robert Lai. 1999. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison Wesley Professional.

Jon Whittle and John Hutchinson. 2011. Mismatches between Industry Practice and Teaching of Model-Driven Software Development. In *Proc. of Models in Software Engineering - Workshops and Symposia at MODELS 2011, Reports and Revised Selected Papers*. Springer, Wellington, New Zealand, 40–47.

Claes Wohlin, Aybüke Aurum, Lefteris Angelis, Laura Phillips, Yvonne Dittrich, Tony Gorschek, Håkan Grahn, Kennet Henningsson, Simon Kågström, Graham Low, Per Rovegard, Piotr Tomaszewski, Christine Van Toorn, and Jeff Winter. 2012. The Success Factors Powering Industry-Academia Collaboration. *IEEE Software* 29, 2 (2012), 67–73.

Online Appendix to: Teaching Software Product Lines: A Snapshot of Current Practices and Challenges[3]

Mathieu Acher, University of Rennes 1, IRISA/Inria, Rennes, France
Roberto E. Lopez-Herrejon, École de Technologie Supérieure, Montreal, Canada
Rick Rabiser, CDL MEVSS, ISSE, Johannes Kepler University, Linz, Austria

A. QUESTIONS OF THE SURVEY ON TEACHING OF SPLS

(1) What is the name and country of your institution?
- (free text)

(2) What is the type of your institution?
[ ] College – teaching focused
[ ] Research focused
[ ] Other: (free text)

(3) What is your department?
[ ] Computer Science
[ ] Software Engineering
[ ] InformationTechnology
[ ] Other disciplines: (free text)

(4) What is your number of years of experience in researching product lines?
(free text)

(5) What is your number of years of experience in teaching product lines?
(free text)

(6) List the primary literature used in the course
[ ] Books: (free text)
[ ] Research Papers: (free text)
[ ] Case Studies: (free text)

(7) Tools used (especially product line and variability tools)
(free text)

(8) Context of the teaching
[ ] Self-contained course
[ ] Part of a course, please specify: (free text)

(9) Audience
[ ] Undergraduate students
[ ] Graduate students
[ ] Industry people

(10) Total lecture time (e.g, 3hours per week * 15 weeks = 45hours / semester)
(free text)

(11) Total practical time (i.e., same as previously: detail the total number of hours dedicated to lab sessions and practical exercises)
(free text)

(12) Length of the course
( ) Semester
( ) Quarter
( ) Other: (free text)

(13) Parts of product line development covered

DOI: 0000001.0000001

[ ] Requirements engineering
[ ] Testing
[ ] Modelling
[ ] Implementation
[ ] Maintenance & Evolution
[ ] Reverse Engineering & SPL Adoption
[ ] Processes
[ ] Other: (free text)

(14) What is the most challenging part for teaching SPL? (e.g., administrative delays, acceptance in the curriculum, lack of material)
(free text)

(15) What is missing to make SPL mainstream? (Suggestions to improve the state of the teaching of SPL.)
(free text)

(16) How can teaching improve/impact research on SPL? (e.g., student participation in case studies, experiments)
(free text)

(17) How can teaching SPL impact industrial practice of SPL?
(free text)

(18) Please add any other comments, concerns, etc. that are relevant for product line/variability teaching
(free text)

B. QUESTIONS OF THE SURVEY ON LEARNING ABOUT SPLS

(1) Where are you from? Please fill the name of your country (nationality)
(free text)

(2) Name of your university/college
(free text)

(3) What program/department are you in?
( ) Computer Science
( ) Software Engineering
( ) Information Technology
( ) Other: (free text)

(4) What's your highest degree?
( ) Bachelor
( ) Master
( ) PhD
( ) No degree
( ) Other degree: (free text)

(5) Do you have at least basic knowledge in the following fields?
[ ] Software Engineering
[ ] Requirements Engineering
[ ] Software Architecture Design
[ ] Reverse Engineering/Software Maintenance
[ ] Software Processes

(6) What's your current job?
[ ] Student
[ ] Working in Industry (please specify the domain in case): (free text)

(7) What did you expect from this course before it started?
(free text)

(8) What did you actually learn from this course?
(free text)

(9) What did you not learn/would have wanted to hear more about?
    (free text)
(10) Please define in your own words what a Software Product Line is
    (free text)
(11) What do you think is the most challenging part of learning about Software Product Line? (examples: hard to perceive the concept of large-scale reuse; requires too much background knowledge)
    (free text)
(12) Suggestions to improve the state of the teaching of Software Product Line
    (free text)
(13) How can the teaching of Software Product Line impact industrial practice?
    (free text)
(14) Any other comments (for example: what did you like it the most, what did you like the least, why?)
    (free text)