Douglas Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.): INFORMATIK 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

Software Product Line and its Products

Bernhard Rumpe [1] Christoph Schulze [1] Johannes Richenhagen [2] Axel Schloßer [2]

Abstract:

Establishing and maintaining a software product line for a series of similar applications is a complex and time-intensive process, which can only pay back its cost if the provided software components achieve a high degree of reuseablity. Furthermore, in many domains, including automotive, stability will always be prioritized over reusability. Integrating a set of existing similar but individually developed products into a reusable product line is often very time-intensive. It is better, but still time-consuming, to feed the result of a new product back into an already existing software product line. Due to resource constraints for the next product, necessary resources are rarely provided. As a consequence, a product line is often endangered by becoming outdated and thus less usable. Therefore, the establishment and maintenance of a software product line needs to be supported by processes and techniques which allow to derive necessary information from running projects with only minor or ideally non-manual effort. In this paper, an agile and semi-automated process to extract and maintain a software product line during the parallel evolvement of several products is proposed. This process is based on PERSIST, an industrially used approach which combines agile techniques and sustainable, long-term architecture development. At the end of this paper, the current status and achievements of PERSIST are discussed, already implemented techniques are evaluated in the context of the automotive powertrain, and further potentials are highlighted.

1 Introduction and Motivation

In the automotive domain, the complexity of software functions and the demanded quality standards, e.g. CMMI, ISO 26262, are still growing, while in the meantime the expected release cycles get shorter and shorter [Br06]. In addition, the vehicle starts to turn into a smart device which is able to interact with its environment and to react autonomously. As a consequence, further aspects, such as security or privacy, receive a higher prioritization [Pr07]. Features, e.g. connectivity, which seem to be of no importance in the automotive domain, may become mandatory for the customer, while main selling points of the past, e.g. driveability, could switch to secondary priority if not connected to the digital world properly. In parallel the complexity of the general required functionality of the automotive powertrain is increased, as more and more standards have to be fulfilled.

[1] Software Engineering, RWTH Aachen, Ahornstraße 55, 52074 Aachen, http://www.se-rwth.de
[2] FEV GmbH, Neuenhofstraße 181, 52078 Aachen, http://www.fev.com

[RSRS15] B. Rumpe, C. Schulze, J. Richenhagen, A. Schloßer:
Agile Synchronization between a Software Product Line and its Products.
In: Informatik 2015, LNI P-246, pp. 1687-1698, Bonner Köllen Verlag, 2015.
www.se-rwth.de/publications

This up-to-date scenario is a great example for the frequently changing and hardly predictable demands in today's automotive industry. Who can ensure that features which are identified as necessary today will be required in a similar form tomorrow? What is the adequate time frame for long-term strategies in the context of shorter and shorter release periods? How can already established software structures be connected to new sub-domains and thereby keep a maximum degree of reusability?

In [RPS14], PERSIST (powertrain control architecture enabling reusable software development for intelligent system tailoring) has been established to give first answers to these questions in the context of an engineering service supplier. This approach introduces agile methods to be able to react flexibly on requirement changes and to reduce the duration of one development cycle and project quality risks due to continuous integration. In addition standards like AUTOSAR and the ISO 25010 have been taken into consideration during the development of PERSIST to ensure a high degree of quality and to be able to comply to requested formats in todays automotive industry. PERSIST focuses on small development cycles in which a small amount of software components are planned, realized and verified. These components are verifed automatically by the nightly build server mostly. The approach was successfully applied first in the context of a two-stage turbocharged gasoline engine [RPS14]. Based on this experience, further applications have been performed [VRP15, Ri15].

Nevertheless, the establishment and maintenance of a Software Product Line (SPL) in the context of the daily work of a supplier has to face several ongoing projects in parallel. Each of these projects have similar demands, but different customers. Often these customers are only interested in the development of a product rather than in the establishment of reusable software components or a complete software product line, at least not in the degree it would be sufficient for the supplier.

As a consequence, an approach is required which allows the project teams to focus on the implementation of the required product, while it is possible to continuously establish and maintain a SPL in parallel with minimized effort. Therefore, it is necessary to further extend the collaboration between Agile Software Development (ASD) and Software Product Line Engineering (SPLE), resulting in Agile Software Product Line Engineering (APLE).

In this paper, we propose such an approach based on the already established fundament of PERSIST. The paper is structured as follows: in Section 2 we give a short overview about SPLE and ASD to be able to give a more detailed overview of the current status in the research field of APLE. The section finishes with summarizing metrics which build the foundation for the proposed approach. Section 3, describes the proposed approach in all detail, while in Section 4, its application is evaluated. Section 5 summarizes related work and in Section 6 open points are discussed and future work is derived.

2 Foundations

SPLE [PBL05] focuses on establishing a reusable platform for a specific domain which allows to derive several customized products in an efficient manner (Figure 1). The SPLE

paradigm divides the development into two separate phases: During the Domain Engineering (DE) phase the most common aspects of the domain are identified and based on this analysis a software platform is derived. During this step, the commonality and variability of the product line are defined. One important step is the definition of a reference architecture which represents the common high-level structure of the product line.

Based on this established platform, domain artifacts are reused and open variation points are bound to extract a specific product during the Application Engineering (AE) phase [PBL05]. To express variability in domain artifacts different annotative, compositional, and transformational variability modeling approaches have been proposed [Sc12].

A SPL can be established either in a proactive, reactive or extractive manner [Kr02]. The proactive approach derives the whole product line from scratch, while the reactive approach derives the product line in an incremental manner. In many cases, the initial product line is derived from a given set of products, therefore an extraction is performed.

ASD unites a set of methods which are build upon the Agile Manifesto [1]: *priorizing individuals and interactions, working software, customer collaboration and responding to change over processes and tools, comprehensive documentation, contract negotiation and following a plan*. This is achieved by performing small development cycles which are driven by related acceptance criteria. Long-term analysis and development plans are avoided as frequent changes are expected. SCRUM [Sc04] or extreme Programming [BA04] are popular software development processes in the context of ASD, while Test Driven Development (TDD) [Be02] and continuous integration are two of the most common agile methods [GPM08].

SPLE and ASD both pursue at the same targets: customer satisfaction and reduced time-to-market [Dí11]. It has already been demonstrated that both approaches are able to achieve the specified goals [CN02a, Li04]. In addition, both approaches suggest to manage fre-

[1] The Agile Manifesto. `http://www.agilemanifesto.org`

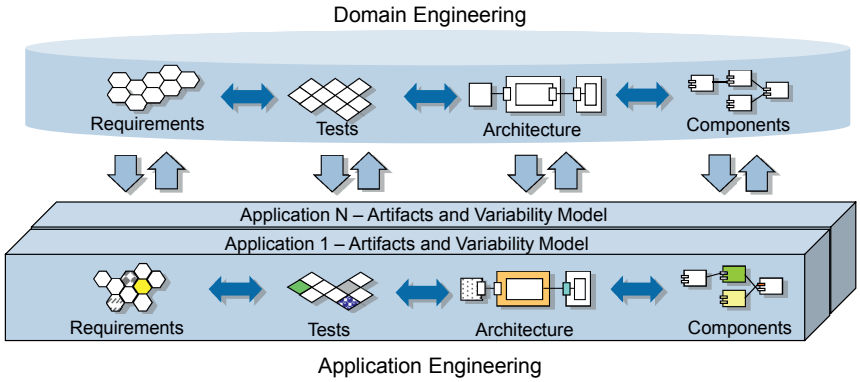

Fig. 1: Domain and application engineering (based on [PBL05]).

quent requirement changes efficiently [TC06]. In the context of APLE, it is tried to establish a hybrid of both approaches. Different researches state conflicts between ASD and SPLE [MRS10, HF08]: the nature of ASD enforces short development cycles with a short planning period, while the establishment of a SPL requires the prediction of upcoming requirements by an intensive requirement analysis phase.

Reducing the long-term investment during the DE phase is one of the main reasons for combining SPLE and ASD. Enforcing the performance of DE before the AE has been identified as very cost-intensive and risky [Dí11]. Establishing and maintaining a SPL leads to significant coordination overhead and slower release cycles [Bo10]. In the context of frequently changing demands, the threat of losing long-term investments becomes even more realistic. For a supplier traditionally dealing with frequent change requests from customers and with having low interest for long-term investments, these aspects are of even higher importance. In addition, [Ca08] highlights the importance of an intensive feedback from AE teams to the DE team. One posibility to avoid long-term integration could be enforcing a short-term integration, which would move the effort to smaller but more frequent system integrations [Bo10].

Very important for an iterative extraction of a product line based on established products is the necessity to identify the most applicable parts of a product for an extraction. Therefore it is inevitable to derive related metrics. In [Be10, BRR10], a set of metrics to measure such an ability is defined. These metrics are based on three levels of component similarity: first, the components' names are compared (extrinsically equal) afterwards their interfaces (syntactically identical) and finally their behavior (semantically identical). While the interface similarity is derived on the basis of ids and signatures only, the behavioral similarity is evaluated by applying given test sequences. In [BRR10], the metrics have been adapted to also measure gradual similarities between components. This kind of relationship model has been applied to identify components which are, to a specific degree, either similar to all other products, just to one or to none. By applying this model, different signatures are matched based on their related output only.

3 Application Engineering focused Software Product Line Development

Recapitulating the statements from Section 2, a SPL development method is required which provides essential feedback for DE during AE, but reduces the additional effort for AE to a minimum. A method which does not require long-term decisions and which keeps the SPL up-to-date and identifies new potential for reusable components is needed.

In the following section, such a method is described which is shown in Figure 2. The coloured activities represent activities which are performed by DE, while the white ones are performed during AE. The main important development artifacts used during this process are the reference architecture, project architecture, component specification, test cases and the component implementation (not shown). In general, the developers from the AE have access to the reference architecture of the SPL, while the developers from DE have

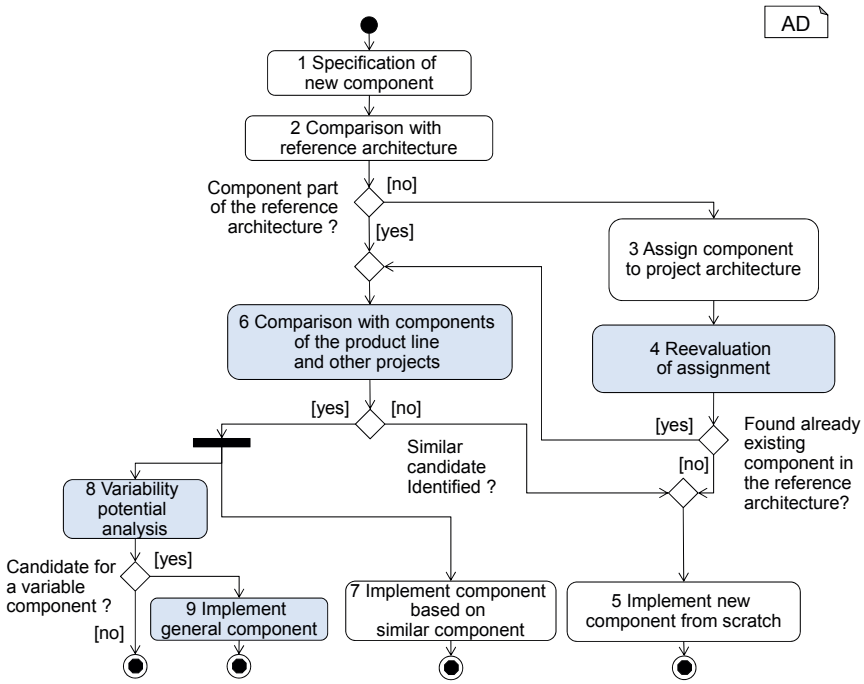

Fig. 2: Method for agile software product line development focused on application domain. (colored = DE activities, white = AE activities)

access to all projects and related development artifacts. In addition, each project has a specific project architecture which is very similar to the reference architecture.

The proposed method follows a component-based AE-first approach: the main item of the software architecture is a component and any specification for a component is first raised during AE. A component will only be considered to be redesigned in a more general way, if it is proven that a corresponding demand is given and a general component is a realistic opportunity within several projects. In addition, AE shall gain most benefit from the already established product line and components which are currently developed in different projects, without being slowed down by the burden of being dependent of generalized components. Next the steps shown in Figure 2 are described in detail, whereby the order defined in Figure 2 is followed.

In the first step, a new component is specified (*1 Specification of new component*). This is done by performing a first draft of the interface with a related functional description. In addition, its position in the software architecture is estimated. This is done by considering the reference architecture of the SPL (*2 Comparison with reference architecture*). In the context of PERSIST, components are hierarchically arranged in a set of compositions. If a suitable component can be identified, which seems to be similar or identical to the specified component, the name and location of the component will be adapted to

the reference architecture. This is the first step, where the SPL supports the development during AE. Complex architectural decisions can often be supported by experiences from the past (which are stored in the reference architecture). In case the component cannot be mapped, a specific position in the project's software architecture needs to be defined (*3 Assign component to project architecture*).

In general, the introduction of new components (components which are not part of the current reference architecture) should be avoided, where possible, to reduce the complexity of the reference architecture. In addition, further evaluations of similarities can only take place efficiently, if components could be mapped to a similar context (name, composition). Therefore, in a second step the decision made in *3 Assign component to project architecture*, is reevaluated by the DE team to avoid any false negatives (*4 Reevaluation of assignment*). Only if both the AE team and DE team cannot map the new component to the reference architecture, a further similarity analysis will not take place. To reduce the effort for AE during step *2 Specification of new component* the main responsibility to avoid false negatives lies with the DE team. If the component cannot be matched the complete implementation will be performed from scratch (*5 Implement new component from scratch*): no benefit can be gained from the SPL but the additional effort for the AE team is also small. Furthermore, the new component is added to the reference architecture after its location has been agreed on (can also be applied after step *4 Reevaluation of assignment*).

If the component specified in the context of the project can be mapped to a component an extrinsical equality [Be10] regarding the reference architecture can be established. This connection cannot only be used to compare the component, which is currently under development, with general components of the SPL but also to compare it with other individual, but extrinsical, equal components from different projects.

During such a comparison (*6 Comparison with components of the product line and other projects*) the most similar component is derived by the DE team by applying a gradual version of the identity relationship model [Be10]:

$$\Delta_{\approx}(c_x, c_y) = \frac{|W|}{|S_{c_x} \cup S_{c_y}|} \qquad \text{with W} = \{s \in S_{c_x} | \exists t \in S_{c_y} : s = t \wedge b_s = b_t\} \tag{1}$$

whereby the syntactical interface set S_{c_j} of component c_j consists of r concrete signatures $s_k = id \times \mathscr{P}(\mathbb{N}, \mathbb{R}, [TYPE], ..\})$ and $b_k : s_k \rightarrow \mathscr{P}(\{\mathbb{N}, \mathbb{R}, [TYPE], ..\})$ prescribes the behavior of the signature s_k [Be10].

To be able to identify semantical similarities automatically, corresponding test cases need to be provided, as suggested in [BRR10]. As consequence a TDD is required to derive similarities as early as possible. If they cannot provided, only structural similarities will be extracted automatically. A semantical analysis has to be performed by experts, contradicting the general goal to reduce the additional effort for the AE team to a minimum. In the best case, the similarity is 100%: the whole component can be reused directly, no additional effort is given. In the worst case, no useful similarity can be identified: the component has to be implemented from scratch (*5 Implement new component from scratch*).

If a candidate with a similarity lower than 100% is identified, this candidate can be used by the AE team to finalize their own specification and the implementation can be based on the provided development artifacts (*7 Implement component based on similar component*). The AE team does not spend any additional effort on implementing a reusable component which is able to fulfil the requirements of both or more similar components. The identified similarities are only used to speed up the development during AE.

In a parallel step (*8 Variability Potential Analysis*) the DE team evaluates if the identified similarities provide enough potential for a general reusable component. Based on the amount of similar components and the degree of similarity on structural and semantical level this decision has to be made (*7 Implement component based on similar component*). If the potential is high enough, the DE team will implement a general component which can then be reused in further projects.

Depending on the actual milestones in the project and the necessary effort to implement the general component, the AE team could also decide to skip their own individual implementation and directly apply the general component. Of course this would further reduce the effort during AE, but in general it is suggested that the necessary time frame or resources are not available. If corresponding expertise and resources are available, the AE team could also implement the general component, therefore performing a DE team task.

The proposed steps ensure that the AE team always specifies component which are as close as possible to the already established reference architecture and related general components. In the best case general components can directly be reused. Additionally all derivations of the reference architecture can automatically be spotted and the degree of variation evaluated. Therefore the potential degeneration of an established product line can be observed continuously and a synchronization between product and product line can be performed iteratively.

4 Evaluation

Since the first application of PERSIST in [RPS14] it has successfully been introduced in 14 different projects (RCP and series development in the context of the automotive power train) . The proposed approach could in part be established at the FEV GmbH. Based on PERSIST which provides a set of standards regarding component and datatype definitions it has already been possible to evaluate potentials and necessary improvements. The described activities are just an extension of PERSIST, no costly adaption of the established process are necessary.

Activities 2, 3, 4 (and of course 1 and 5) are already well established, while for activity 6 and 8 the necessary automatization to efficienty perform these steps is currently missing (see Figure 2). General components, which are used in several projects, have already been established, but always based on intensive manual reviews or during a proactive approach. The possibility to investigate a given reference architecture during the specification of a new component is not noticed as an additional effort. Instead the reference architecture provides helpful information to perform architectural decisions.

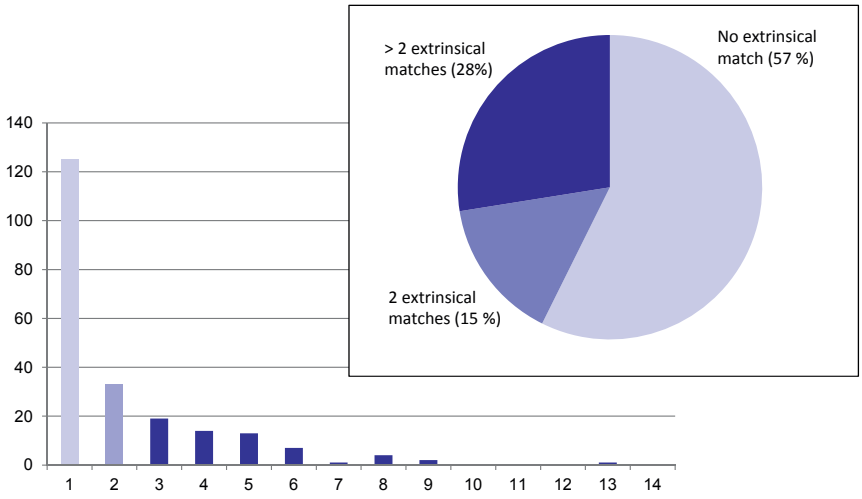

Fig. 3: Amount of components with *n* extrinsically identical components in several projects.

The comparison made in activity 4 is supported by a nightly build script which compares the different project architectures with the reference architecture. If a component cannot be extrinsically mapped to the reference architecture, a corresponding report will be generated. Based on this report, the DE team can reevaluate the architectural decision and inform the corresponding project, if necessary. This way, the reference architecture can semiautomatically be synchronized. Necessary time-intensive discussions to identify the right location for a new component could not be reduced, but the effort to spot potential deviations has been significantly minimized. Furthermore, it can be evaluated automatically how often a component of the reference architecture occurs in the different projects (how often an extrinsically identical component is given). This information provides the access point for further structural and semantical evaluations (Activity 6). In addition it provides a first rough analysis of potential reuse and a good starting point for every implementation performed by the AE team.

The results of the extrinsical comparison are shown in Figure 3. In the current status the reference architecture consists of 219 components, whereas 125 (57%) components have no extrisincally equal match. 94 (43%) components of the reference architecture are part of more than one project and 61 (28%) of these components are also part of at least three projects. To check the feasibility of the relationship model proposed in [Be10], all components which occur at least in 5 projects have been evaluated regarding their structural similarities. In most of cases, none or very small structural similarities could be identified. A comparison based on the concrete signature, as defined in [Be10], appears not to be sufficient. In many cases, the signatures are not identical but still very similar. During the manual reviews of the components interfaces, many signatures could be identified which only differed in their specific representation: often only different units, similar ranges or

other levels of abstractions result in changed signal names. In addition a manual analysis of different interfaces has been identified as very time-intensive and still error-prone.

5 Related Work

In a reactive manner Salion developed an initial set of systems to derive commonalities and identify variabilities [CN02b] in a second step. This case study already provides first input on a possible collaboration of ASD and SPLE: variable requirements have often been realized by custom solutions instead of generic components [MRS10]. Refactorings have been performed selectively, only smaller adaption on the product line scope, instead of larger changes are performed [MRS10]. In addition, clone detection has been used to identify candidates for an extraction. Still these clone detection mechanisms are not used to identity reuseability potentials during the AE phase.

[Zh11] describes an incremental and iterative reengineering approach which focuses on a component-wise establishment of a product line. This idea is based on a suggestion from [SV02] which proposed to either establish a product line as a whole or component-wise. In [Zh11], based on the developer's expertise, components with a high level of variability and expected intensive maintenance or adaption effort are selected first to be integrated into the SPL. By comparing the maturity level and the degree of similarity of same components in other projects, the reference component can be identified. The approach has been applied in a specific project in which a product line based on 5 products was established. During the incremental establishment of the product line the integration effort and faults report trend significantly decreased, which resulted in positive accumulated profit after the second product had been derived. In comparison to the provided approach no explicit metrics have been defined to measure similarities between components or to analyze the strategical importance for the SPL. Therefore a possible automatization of this task is not discussed. In addition an initial comparison of components through an additional mapping to the reference architecture at the beginning of the development cycle is not part of the process. Potentials of reuseability are only identified due to the reengineering of the software product line. Even component-level automated test-suites are provided to ensure a proper reengineering, these artifacts are not used for any kind of similarity analysis.

In [Ca06] incremental design [BA04] and the planning game [Hu05] are applied to PuLSE-I, a reuse centric application engineering approach, to complement the benefits of SPLE and ASD. In a later step the planning game has been adapted to provide feedback between the domain and the application engineering team [Ca08]. In contrast the proposed approach focuses on efficient similarity analysis in the AE phase due to agile techniques like TDD and continious integration. An introduction of the planning game is currently not considered, but maybe will be applied in the future.

In [GM10, GPM08], a test-driven reactive SPLE approach is described and evaluated. In this approach, new variation points are introduced by using the test artifacts as a starting point. Even though the authors mention in [GPM08] the possibility of comparing acceptance tests to evaluate similarities between given systems and new requirements, this idea is not evaluated further.

Compositional software product lines as described in [Bo10] allow to move all necessary coordination overhead to the architecture level: each team decides on its own which new functionality needs to be implemented and ensures that each new version provides a backward compatibility regarding its interface. While this approach is also architecture-driven, the identification of reuseability potentials is performed individually and is not supported by standardized metrics.

In [Ru12, Ru04] a set of agile methods is proposed to define complex systems based on UML/P, a profile of UML. These methods build a foundation for agile model-driven development with focus on flexibility and customer satisfaction [Ru12].

In [Ho12], a reactive and model-driven approach is suggested, which focuses on the application of reusability in parallel to the project-driven development. Already existing modules are optimized stepwise, as the general introduction of a software product line has been identified as a challenging task.

6 Conclusion

The proposed approach provides the possibility of realizing the project-driven implementation work directly in the project (AE) without loosing the benefits of a corresponding software product line. Each project team can benefit from the established reference architecture to speed up architectural decisions, while the similarity analysis performed during DE can provide additional foundations for the actual implementation. In the long-term, general components can be derived whose reusability potential have been proven through several projects. Furthermore the Activity *4 Reevaluation of assignment* can be fully or semi automated, if adequate information is provided. More complex evaluations for the activities *6 Comparison with components of the product line and other projects* and *8 Variability potential analysis* are currently not automated and require some improvements regarding their granularity to establish a quality similar to a manual review. Hence, a more detailed relationship model and an automated evaluation of semantical similarities based on test cases is planned for the future. Regarding the current status the potential of the suggested approach could not be evaluated totally, but necessary additional modifications are currently developed to be able to perform a more detailed analysis in the future. Moreover, it is necessary to provide the AE team all already established data, such as signals, component definitions or nightly build reports, in an adequate and stable manner to increase the acceptance of the product line. Therefore, it is planned to foster PERSIST with a database supporting signal and model management.

References

[BA04] Beck, Kent; Andres, Cynthia: Extreme Programming Explained: Embrace Change. Addison-Wesley, 2nd edition, November 2004.

[Be02] Beck, Kent: Test-Driven Development: By Example. Addison-Wesley Professional, 1 edition, November 2002.

[Be10] Berger, Christian; Rendel, Holger; Rumpe, Bernhard; Busse, Carsten; Jablonski, Thorsten; Wolf, Fabian: Product Line Metrics for Legacy Software in Practice. In: Proceedings of the 14th International Software Product Line Conference (SPLC 2010). volume 2, 2010.

[Bo10] Bosch, J.: Toward Compositional Software Product Lines. Software, IEEE, 27(3):29–34, May 2010.

[Br06] Broy, Manfred: Challenges in Automotive Software Engineering. In: Proceedings of the 28th International Conference on Software Engineering. ICSE '06, ACM, New York, NY, USA, pp. 33–42, 2006.

[BRR10] Berger, Christian; Rendel, Holger; Rumpe, Bernhard: Measuring the Ability to Form a Product Line from Existing Products. In: Variability Modelling of Software-intensive Systems (VaMos). 2010.

[Ca06] Carbon, Ralf; Lindvall, Mikael; Muthig, Dirk; Costa, Patricia: Integrating Product Line Engineering and Agile Methods: Flexible Design Up-front vs. Incremental Design. In: Workshop on Agile Product Line Engineering. 2006.

[Ca08] Carbon, R.; Knodel, Jens; Muthig, Dirk; Meier, G.: Providing Feedback from Application to Family Engineering - The Product Line Planning Game at the Testo AG. In: Software Product Line Conference, 2008. SPLC '08. 12th International. pp. 180–189, Sept 2008.

[CN02a] Clements, P.; Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.

[CN02b] Clements, Paul; Northrop, Linda: Salion, Inc.: A Software Product Line Case Study. Technical Report CMU/SEI-2002-TR-038, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2002.

[Dí11] Díaz, Jessica; Pérez, Jennifer; Alarcón, Pedro P.; Garbajosa, Juan: Agile Product Line Engineering - A Systematic Literature Review. Software: Practice and Experience, 41(8):921–941, July 2011.

[GM10] Ghanam, Yaser; Maurer, Frank: Extreme Product Line Engineering - Refactoring for Variability: A Test-Driven Approach. In (Sillitti, Alberto; Martin, Angela; Wang, Xiaofeng; Whitworth, Elizabeth, eds): Agile Processes in Software Engineering and Extreme Programming, volume 48 of Lecture Notes in Business Information Processing, pp. 43–57. Springer Berlin Heidelberg, 2010.

[GPM08] Ghanam, Yaser; Park, Shelly; Maurer, Frank: A Test-Driven Approach to Establishing & Managing Agile Product Lines. In: SPLiT 2008–Fifth International Workshop on Software Product Line Testing. p. 46, 2008.

[HF08] Hanssen, Geir K.; Fígri, Tor E.: Process Fusion: An Industrial Case Study on Agile Software Product Line Engineering. Journal of Systems and Software, 81(6):843–854, June 2008.

[Ho12] Hopp, Christian; Rendel, Holger; Rumpe, Bernhard; Wolf, Fabian: Einführung eines Produktlinienansatzes in die automotive Softwareentwicklung am Beispiel von Steuergerätesoftware. In: Software Engineering 2012: Fachtagung des GI-Fachbereichs Softwaretechnik. volume 198 of LNI, Berlin, Deutschland, pp. 181–192, 2012.

[Hu05] Hunt, John: Agile Software Construction. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Kr02] Krueger, C.: Eliminating the Adoption Barrier. Software, IEEE, 19(4):29–31, 2002.

[Li04] Lindvall, M.; Muthig, Dirk; Dagnino, A.; Wallin, C.; Stupperich, M.; Kiefer, D.; May, J.; Kahkonen, T.: Agile Software Development in Large Organizations. Computer, 37(12):26–34, Dec 2004.

[MRS10] Mohan, Kannan; Ramesh, Balasubramaniam; Sugumaran, Vijayan: Integrating Software Product Line Engineering and Agile Development. Software, IEEE, 27(3):48–55, 2010.

[PBL05] Pohl, Klaus; Böckle, Günter; Linden, Frank J. van der: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Pr07] Pretschner, Alexander; Broy, Manfred; Krüger, Ingolf H.; Stauner, Thomas: Software Engineering for Automotive Systems: A Roadmap. In: 2007 Future of Software Engineering. FOSE '07, IEEE Computer Society, Washington, DC, USA, pp. 55–71, 2007.

[Ri15] Richenhagen, Johannes; Venkitachalam, Hariharan; Schloßer, Axel; Pischinger, Stefan: PERSIST — a scalable software architecture for the control of various automotive hybrid topologies. Technical report, SAE Technical Paper, 2015.

[RPS14] Richenhagen, Johannes; Pischinger, Stefan; Schloßer, Axel: PERSIST—A Flexible and Automatically Verifiable Software Architecture for the Automotive Powertrain. Journal of Electrical Engineering, 2:108–115, 2014.

[Ru04] Rumpe, Bernhard: Agile Modeling with the UML. In (Wirsing, M.; Knapp, A.; Balsamo, S., eds): Proceedings of the Radical Innovations of Software and Systems Engineering in the Future. 9th International Workshop (RISSEF'02). volume 2941 of LNCS, Springer, Venice, Italy, pp. 297–309, October 2004.

[Ru12] Rumpe, Bernhard: Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring. Springer Berlin, 2te edition, Juni 2012.

[Sc04] Schwaber, Ken: Agile Project Management with Scrum. Microsoft Press, Redmond, WA, USA, 2004.

[Sc12] Schaefer, Ina; Rabiser, Rick; Clarke, Dave; Bettini, Lorenzo; Benavides, David; Botterweck, Goetz; Pathak, Animesh; Trujillo, Salvador; Villela, Karina: Software Diversity: State of the Art and Perspectives. International Journal on Software Tools for Technology Transfer, 14(5):477–495, 2012.

[SV02] Schmid, Klaus; Verlage, Martin: The Economic Impact of Product Line Adoption and Evolution. IEEE software, 19(4):50–57, 2002.

[TC06] Tian, Kun; Cooper, Kendra: Agile and Software Product Line Methods: Are They So Different. In: 1st International Workshop on Agile Product Line Engineering. 2006.

[VRP15] Venkitachalam, Hariharan; Richenhagen, Johannes; Pischinger, Stefan: A generic control software architecture for Battery Management Systems. Technical report, SAE Technical Paper, 2015.

[Zh11] Zhang, Gang; Shen, Liwei; Peng, Xin; Xing, Zhenchang; Zhao, Wenyun: Incremental and Iterative Reengineering Towards Software Product Line: An Industrial Case Study. In: Software Maintenance (ICSM), 2011 27th IEEE International Conference on. pp. 418–427, Sept 2011.