A concrete product derivation in software product line engineering: a practical approach

Karam Mustafa Ignaim

Software Engineering Department,
Al-Balqa Applied University,
Al Salt, Jordan
Email: karam.ignaim@bau.edu.jo

Khalid Alkharabsheh*

Software Engineering Department,
Al-Balqa Applied University,
Al Salt, Jordan
Email: khalidkh@bau.edu.jo
*Corresponding author

André L. Ferreira

Bosch Car Multimedia Portugal,
Braga, Portugal
Email: Andre.Ferreira2@pt.bosch.com

João M. Fernandes

Universidade do Minho,
Braga, Portugal
Email: jmf@di.uminho.pt

Abstract: Software Product Lines enable the development of a perfect family of products by reusing shared assets in a systematic manner. Product derivation is a critical activity in software product line engineering and one of the most pressing issues that a software product line must address. This work introduces an approach for automating the derivation of a product from a software product line. The software product line is part of a product family that evolved from a non-structured approach to managing variability. The automated derivation approach relies on product configurations and the refactoring of feature models. The approach was deployed and evaluated in the automotive domain using a real-world software product line. The outcome demonstrates that the approach generates a product in an automated and successful manner.

Keywords: software product lines; product derivation; feature models; product configuration; refactoring.

Reference to this paper should be made as follows: Ignaim, K.M., Alkharabsheh, K., Ferreira, A.L. and Fernandes, J.M. (2022) 'A concrete product derivation in software product line engineering: a practical approach', *Int. J. Computer Applications in Technology*, Vol. 70, Nos. 3/4, pp.225–232.

Biographical notes: Karam Mustafa Ignaim received her BSc degree in Information Technology from Al Balqa Applied University, Jordan, the MSc degree in Computer Science from Al Balqa Applied University, Jordan and the PhD degree in Software Engineering from the University of Minho, Portugal. Her research interests include Software Product Lines, Feature Modelling, Software Reuse and Software Maintenance and Evolution.

Khalid Alkharabsheh is an Assistant Professor at Al-Balqa Applied University, Jordan. He received his PhD degree in Software Engineering from the University of Santiago de Compostela in Galicia, Spain. His research interests include machine learning, big data, software quality, empirical software engineering, software validation and verification, design smell detection and object-oriented language.

André L. Ferreira is a Senior Engineering Manager for Product Development at Cross-Domain Computing Solutions. He received the PhD degree in Software Engineering. Also, he is an R&D Engineer at Bosch Car Multimedia Portugal S.A. for the Automotive Segment. He is invited as an Assistant Professor in the Department of Informatics at the University of Minho.

João M. Fernandes is Full Professor at Universidade do Minho, Portugal. He conducts his research activities in software engineering, with a special interest in software modelling, requirements engineering and software business. He is the main author of the book '*Requirements in Engineering Projects*', Springer in 2016. He has been involved in the organisation of various international events, including *ACSD 2003*, *DIPES 2006*, *GTTSE 2009*, *PETRI NETS 2010*, *ACSD 2010*, the *MOMPES Workshops Series (2003–2012)* and *ICSOB 2015*.

1 Introduction

Software Product Lines focus on supporting the processes related to developing a full family of products (Clements and Northrop, 2001; Deelstra et al., 2005). Corporations consistently report a considerable refinement in terms of amount of productivity, product quality, time required for marketing and the client satisfaction due to using software product lines (Ghanam and Maurer, 2009; Alkharabsheh et al., 2018). A product is a single member of a product family with a collection of artifacts that implement its features (Van der Linden et al., 2007). During application engineering, an individual product of a family is created based on software product line assets to address a specific customer need within a market segment (Ghanam and Maurer, 2009, AL-Msiédeen et al., 2013; Alves et al., 2010).

However, it is popular industrial practice to extract the initial product from a software product line, then add and adapt features to meet the needs of individual customers (Gao and Gu, 2021; Azar et al., 2020). These modifications are then incorporated back into the original software product line (Van der Linden et al., 2007; Hinterreiter et al., 2018). Companies like Philips, Bosch and Nokia can now build customer-specific functionality with minimal effort, resulting in important enhancements in marketing time, cost, productivity and quality (Clements and Northrop, 2001; Abbasi et al., 2022). The variability feature, which is explicitly addressed during the development process, distinguishes software product line engineering from single product development. Products are created by resolving variability to implement the functionality of customer-specific, which is typically accomplished through the use of a product derivation process.

Many businesses attempt to leverage software product line engineering while maintaining a large number of product variants in their product lines. These huge product lines include a lot of products (hundreds) with a high degree of variation and numerous configurations, particularly in the automotive domain (Steger et al., 2004; Maccari and Heie, 2005). Because the majority of product development work is done by hand, the systematic derivation of products is considerably difficult, time-consuming and error-prone (Deelstra et al., 2005). Several studies (Alkharabsheh et al., 2018; Botterweck et al., 2009; Monestel et al., 2002; Thao et al., 2008; Rabiser et al., 2011; Camacho et al., 2021) and tools (Van Ommering et al., 2000; Tryggeseth et al., 1995; Pohl et al., 2005; Sinnema et al., 2006) for software product line product derivation are focused on variability management, offering assistance to describe and represent variability between software products. Almost none of them provide a real-world solution, and they do not consider industrial case studies when developing and evaluating product derivation approaches. Although there have been studies in the literature that address automatic product derivation in software product line, there is a dearth of studies that address industrial case studies in the automotive domain. The significance of this work is summed up by applying the automatic product derivation technique to a real-world case study in the automotive domain.

We present in this paper a practical approach and research tool for assisting automated product derivation in software product lines. The specific technique is a subset of a larger one described in our previous work, which supports feature modelling and traceability of software product lines (Ignaim, 2021). Our approach is based on the feature model refactoring notations presented in Alves et al. (2006), which allow us to automatically derive a given product from a set of all possible products and then specify its code. Product configurations are used in the derivation (i.e., combinations of features).

The following are the main contributions of this work:

- Based on the feature model, we propose a novel practical approach for automated product derivation in software product line.
- We develop a research tool called 'FriendlyMapper', which allow software engineers to choose product feature combinations.
- In a real-world case study in the automotive domain, we evaluate the proposed approach.

The remainder of the paper is organised as: Section 2 describes related studies. Section 3 presents the proposed approach that was followed to achieve the main objective of the paper. Section 4 explains the evaluation process of the practical approach while the conclusions and future works are discussed in Section 5.

2 Related work

This section discusses software product line studies related to concrete product derivation. Monestel et al. (2002) proposed architectural constraints for the software product line to determine the association among various software architecture factors. They used OCL constraints in the context of UML to create design patterns. These constraints are employed in the software product line to represent architectural variability. A case study from the Mercure project was used to evaluate the approach. In O'Leary et al. (2002), a novel framework for the process was developed, which includes significant tasks that stakeholders of software product lines should perform during product development.

The framework's foundation is built on academia and industry. In academic circles, the framework identifies areas of uncertainty and assists in determining the remaining challenges. As a result, we prioritise the addition of missing parts or additional detail that may be required for a specific purpose. In industry, on the other hand, the proposed framework will provide corporations and organisations with a structured approach to product derivation, making the process more manageable and controlled. In Botterweck et al. (2009), a novel approach to architecture product derivation based on selectively copying elements from software product lines using a specific product feature structure was also introduced. The approach focuses on determining a product's high-level architecture. Following that, O'Leary et al. (2010) and O'Leary et al. (2012) presented a blueprint for software product line derivation based on the agile process model. The proposed approach is known as 'Pro-PD', and it is appropriate for both small and large organisations in terms of upfront investment and balancing formality and agility. The approach's focal point is the fundamental tasks and roles required to derive the product from software product line. An industry case study was used to evaluate the model. Another piece of work done by Lee and Kang (2010) was to extract important information from the product context in order to assist software engineers in the process of feature selection during the software development process. As a result, this data will be used to determine the best product configuration. The useful data will be formulated as constraints or criteria imposed on the selected features.

The OntoAD framework, which is an automatic ontology-based approach for product architecture derivation from software product line architecture, has been introduced in Duran-Limon et al. (2015). For specifying the product architecture and derivation activities, respectively, a language-independent model and model-driven engineering are used to generate the product architecture and derivation activities. Another study conducted by Lahiani and Bennouar (2017) presented a method based on feature-architecture mapping to automate feature derivation based on the source code of a specific transformation language implemented in Java. The proposed framework represents the software product line with a collection of joint models and automatically derives the applicable products from the model. Particle Swarm Optimisation was recently used by Afzal et al. (2020) to minimise disagreements in the designed software product line. The work being done is aimed at generating a consistent configuration from the available feature model. The proposed approach, called o-SPLIT, was evaluated using three cases that involved software product lines, an online tool for Benchmarking and Testing the analysis (BeTTy) and standard ERP. The results show that the technique optimises the software product line's configuration. Furthermore, particle swarm optimisation has been turned into a tool.

3 Proposed approach

The practical approach is based on real-world limitations that were determined in the automotive product family that the authors investigated at Bosch Car Multimedia S.A. The issue is related to a lengthy, repetitive, unstructured process for managing variability. Furthermore, it is based on the ideas, conclusions and results of the previously published review study in Rabiser et al. (2011). The process of creating a software product from the efficient fundamental set of core properties that comprise the architecture is known as product derivation (i.e., sub-systems and components in our case), design (i.e., features in our case), and reusable code (i.e., feature-related code fragments in our case), is a central practice in software product line engineering (Van Gurp and Prehofer, 2006; Bolander and Clements, 2021). The product derivation process proposed by our approach entails selecting, eliminating, extending, and, in some cases, even modifying the current feature model and traceability links (from the features to the code). Both have been described previously (Ignaim and Fernandes, 2019; Ignaim, 2021) and are used in the derived products.

The steps followed for deriving a product from the software product line based on its configurations are described in the next few paragraphs (i.e., features that are a combination of the product or a set of features of the product). A set of features defines a concrete product of the 'Classical Sensor Variants Family' (CSVF) software product line. These features are chosen and removed from the current feature model based on the product's feature combinations and a feature mapping of each feature to the feature-related code fragments [16].

Table 1 displays the feature sets (the optional features) of each CSVF resulting software product line product. We propose that common features such as {'diagnosis', 'monitoring', 'message', 'transmit', 'receive', 'value_1', 'value_2' and signals (_1,_2,_3,_5,_6,_7,_8,_11,_12)} be included in the feature set of each product for reasons. It is assumed that a new customer requests a product called ProductDerived, and the Classical Sensor Development Team (CSDT) writes the requirement document for this product according to the sensor requirements and the client's needs. Using our approach, we perform a product derivation from the CSVF resulting software product line (see Table 1) by going through the following three steps:

Table 1 Products of the CSVF

product no.	*product configurations*
Product 1	layoutR_1, layoutT_1, flag_1, algorithm_1
Product 2	layoutR_2, layoutT_2, flag_2, algorithm_2, signal_4, signal_9
Product 3	calculation, layoutR_2, layoutT_2, flag_2, algorithm_2, value_3, signal_13, signal_9, signal_10
Product 4	identification, flag_1, layoutR_1, layoutT_1, algorithm_1, value_3, signal_10

In *step 1*, the project technical manager selects the required *ProductDerived* features from the current feature model (derived from our previous work (Ignaim and Fernandes, 2019)). 'diagnosis', 'monitoring', 'calculation', 'message',

'transmit' with the layoutT_1 feature, and 'receive' with the layoutR_1 feature are all supported by the *ProductDerived*. It also supports signals (_1, _2, _3, _5, _6, _7, _8, _11, _12) as well as the optional signals (_4,_9,_10,_13) and it supports 'flag_1' from the alternative group flags (_1,_2,_3). Algorithm_1 is used by the *ProductDerived*. Ultimately, signal_12 of the product will be a group feature involves the mutual features value_1 and value_2, as well as value_3 feature which is the optional one. In this section, we always consider the 'signal_transmit' and 'signal_ receive' features to be label features, and they are included by default in feature sets if their parents are present. Figure 1 depicts the ongoing feature model as identified by the project director with the *ProductDerived* needed features (black filled circles in the left-upper of the rectangles).

Step 2 focuses on defining the product that will be derived through choosing and removing features from the current feature model. Figure 2 depicts the feature combination of a single product (i.e., ProductDerived). Step 2 uses the feature model refactoring technique (Ignaim and Fernandes, 2019) to eliminate (remove), add or modify features of the current feature model, which propagates features change from the feature modelling level to the software product line level.

Finally, in *step 3*, in order to acquire the (ProductDerived), which is the resulting product implementation, we have used a tool named friendlyMapper tool[1] (Ignaim, 2021). The derivation process is based on mapping every feature of the feature combination of the *ProductDerived* to feature-related-code routines in the implementation code. As can be seen in Figure 3, the selection of variant derivation is made from the menu mapping context when clicking (right clicking) the 'CSVF' second level item of the traceability tree. This selection will lead to the variant derivation screen as shown in Figure 4.

Figure 1 The required features of the derived product are highlighted in the current feature model

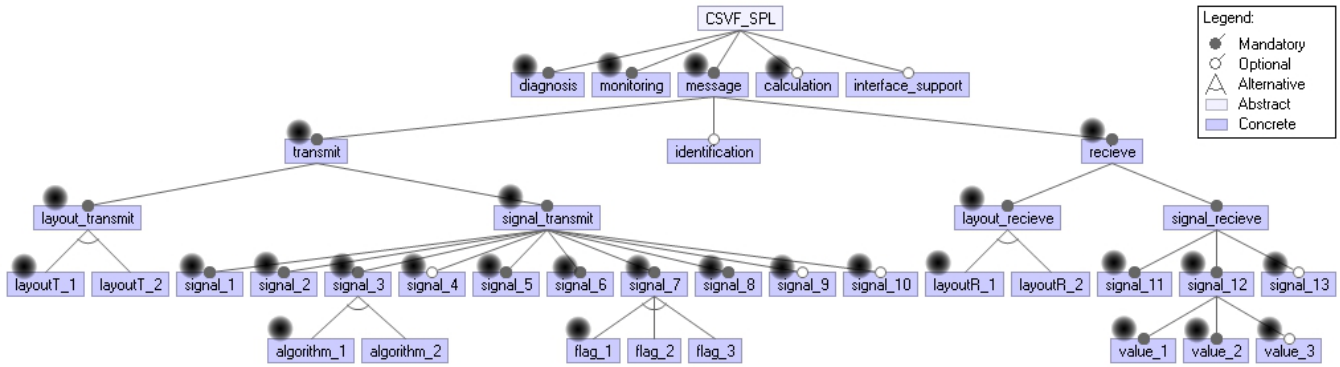


Figure 2 The derived product's feature combination

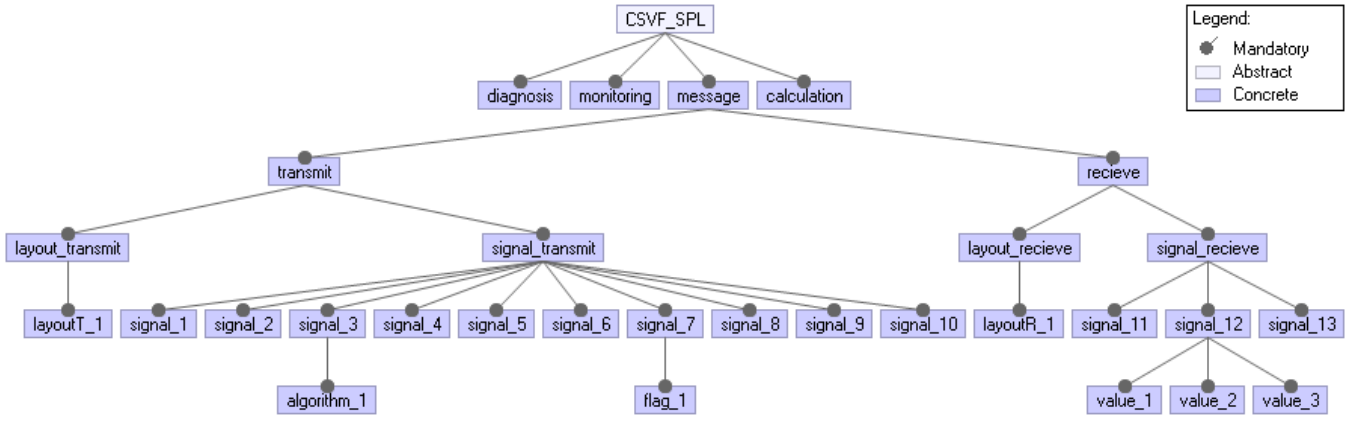


Figure 3 The 'variant derivation' selection from the context menu

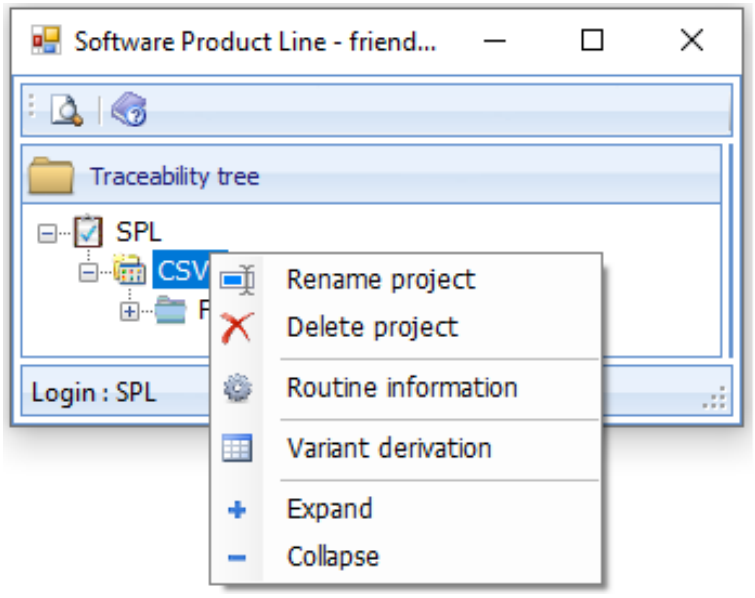


Figure 4 The FriendlyMapper tool's 'variant derivation' screen

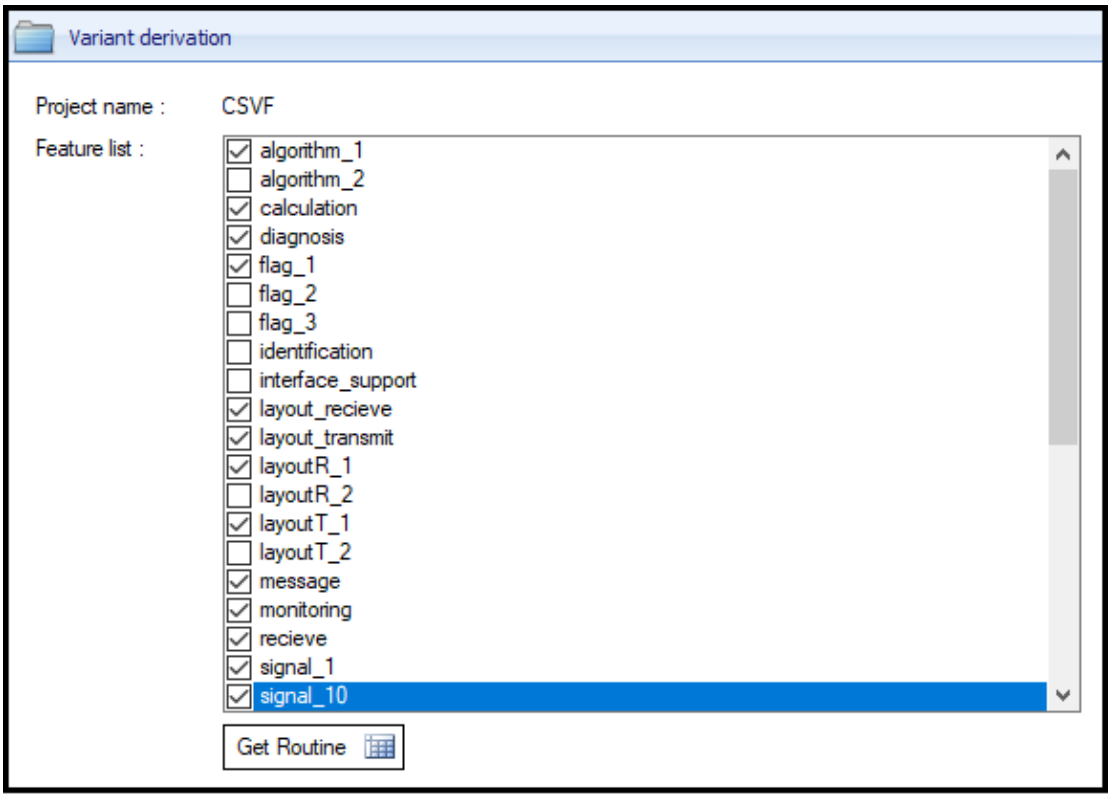


Figure 4 depicts the FriendlyMapper tool's 'variant derivation' screen, which allows software engineers to select a product's feature combinations. Other screens that perform additional software product line-related functionalities are presented in the Ignaim (2021). Figure 4 presents the chosen features (i.e., the *ProductDerived* features combination) and eliminated features from the traceability tree's features list and the 'Get routine' button. Once software engineers click 'this button', the tool automatically presents the resulting product implementation. Figure 5 depicts an example for derivation process of the *ProductDerived*. It was created by mapping each feature of the *ProductDerived* feature combination to feature-related-code fragments of the code, which define the ProductDerived implementation as a whole. For readability reasons, Figure 5 presents a partial view of each feature and its related routines in the *ProductDerived*.

Figure 5 A summary of each feature and its associated routines in the ProductDerived

4 Evaluation

To evaluate the proposed approach, we used a product (Product 4) chosen from the CSVF's available products, resulting in the software product line shown in Table 1. The suggested method is then used to derive the selected product, which is referred to as '*ProductDerivedEval*'. Following that, we compared the *ProductDerivedEval* feature combination to the feature combination of each product in the CSVF resulting software product line (see Table 1). Finally, we examined the derived product's feature combination *ProductDerivedEval* in the following cases:

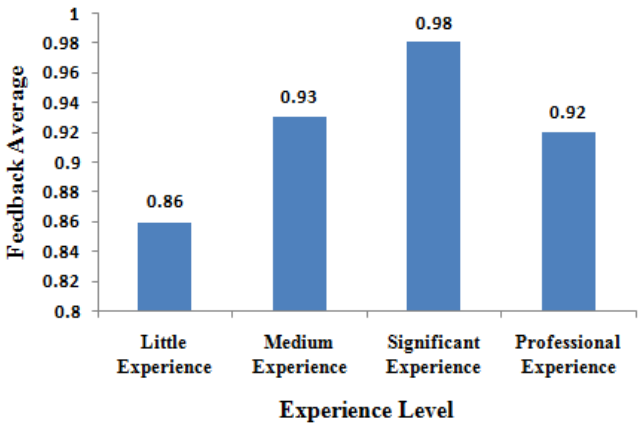
- *Case 1*: A match between the '*ProductDerivedEval*' feature combinations and the feature combinations of one CSVF product, resulting in the software product line (Product 4; see Table 1).
- *Case 2*: The *ProductDerivedEval* set of features is a subset of the set of current feature models derived by our approach.

Given that the result matches one of the cases (Case 1 and Case 2) or both, we can add evidence to the correctness of the product derivation solution presented in this work.

Results: We successfully derived one specific product (see Figure 4) of the CSVF resulting software product line that was not previously supported by the family (ProductDerived) as a result of matching one of the cases or both, during the steps of the approach presented in the preceding section, using product configurations and refactoring of the feature model. Moreover, we have derived another particular product (productDerivedEval) that is already a member of the CSVF resulting software product line (Product 4) as a side effect of evaluating our approach.

Figure 6 presents the result of the survey, which was performed as a small case study for the software developers of CSDT at Bosch Car Multimedia S.A. The developers used our approach to derive a product upon a new customer request, and then they were asked questions referring to the CSDT feedback regarding the proposed approach. The chart shows the average of the positive feedback for the questions presented for each developer on the target team. The CSDT was made up of eight software developers. The graph in the figure shows that seven of them had different levels of experience. Approximately 0.88% (7 out of 8) of the team, including the project manager, highly recommended adopting the proposed approach, which represents 92%. Those software developers have a 'medium, significant and professional level of experience'. Also, all the software developers with a 'little level of experience' prefer to use our approach. With a low percentage difference between them and the software developers at other levels, they reach near 86%.

Figure 6 The relationships between each software developer's experience and the average of positive feedback



Threats to validity: The issues listed below summarise the main threats to the validity: The first is a broadening of the results. Only one product family in the automotive domain was evaluated using the proposed method. Another risk is the evaluation process, which uses only one type of product. To counteract this risk, we intend to replicate the work using additional automotive-related products.

5 Conclusion

This work introduces an approach for automatically deriving a product from a software product line. The software product line evolved from a family of products that began with a non-structured approach to managing variability. The proposed derivation technique is founded on the use of product configurations and feature model refactoring. The method was implemented and tested on a real-world software product line in the automotive domain. The outcome demonstrates that the approach generates a product in an automated and successful manner. In recent decades, the future has required the development of more software systems as a software product line. Thus, the multi-layered architecture will be more widely used. One of the major challenges in the domain will be modelling the variability of existing software product lines and automating configuration. In the future, we plan to make the proposed method more general so that it can be improved and used in more places. To do this, we will include the recent feature models and apply the approach using large case studies from different industry domains.

Acknowledgement

This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

References

Abbasi, T., Hafeez, Y., Asghar, S., Hussain, S., Yang, S. and Ali, S. (2022) 'Towards a component-based system model to improve the quality of highly configurable systems', *PeerJ Computer Science*, Vol. 8.

Afzal, U., Mahmood, T., Khan, A.H., Jan, S., Ur Rasool, R., Qamar, A.M. and Khan, R.U. (2020) 'Feature selection optimization in software product lines', *IEEE Access*, Vol. 8, pp.160231–160250.

Alkharabsheh, K., Crespo, Y., Manso, E. and Taboada, J.A. (2018) 'Software design smell detection: a systematic mapping study', *Software Quality Journal*. Doi: 10.1007/s11219-018-9424-8.

AL-Msiédeen, R.F., Seriai, A., Huchard, M., Urtado, C., Vauttier, S. and Salman, H.E. (2013) 'Feature location in a collection of software product variants using formal concept analysis', *Proceedings of the 13th International Conference on Software Reuse (ICSR'13)*, Springer, pp.302–307. Doi: 10.1007/978-3-642-38977-1.

Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P. and Lucena, C. (2006) 'Refactoring product lines. (2006) '*Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE'06)*, pp.201–210. Doi: 10.1145/1173706.1173737.

Alves, V., Niu, N., Alves, C. and Valença, G. (2010) 'Requirements engineering for software product lines: a systematic literature review', *Information and Software Technology*, Vol. 52, No. 8, pp.806–820. Doi: 10.1016/j.infsof.2010.03.014.

Azar, A.T., Anter, A.M. and Fouad, K.M. (2020) 'Intelligent system for feature selection based on rough set and chaotic binary grey wolf optimisation', *International Journal of Computer Applications in Technology*, Vol. 63, Nos. 1/2, pp.4–24.

Bolander, W.J. and Clements, P.C. (2021) 'Key issues of organizational structure and processes with feature-based product line engineering', *INSIGHT*, Vol. 24, No. 1, pp.42–46.

Botterweck, G., Lee, K. and Thiel, S. (2009) 'Automating product derivation in software product line engineering', *Software Engineering*, pp.177–182.

Camacho, M.C., Álvarez, F., Collazos, C., Leger, P., Bermúdez, J.D. and Hurtado, J.A. (2021) 'A collaborative method for scoping software product lines: a case study in a small software company', *Applied Sciences*, Vol. 11, No. 15. Doi: 10.3390/app11156820.

Clements, P.C. and Northrop, L. (2001) *Software Product Lines: Practices and Patterns*, Addison-Wesley.

Deelstra, S., Sinnema, M. and Bosch, J. (2005) 'Product derivation in software product families: a case study', *Journal of Systems and Software*, Vol. 74, No. 2, pp.173–194. Doi: 10.1016/j.jss.2003.11.012.

Duran-Limon, H.A., Garcia-Rios, C.A., Castillo-Barrera, F.E. and Capilla, R. (2015) 'An ontology-based product architecture derivation approach', *IEEE Transactions on Software Engineering*, Vol. 41, No. 12, pp.1153–1168.

Gao, J. and Gu, Y. (2021) 'Feature matching for multi-beam sonar image sequence using KD-tree and knn search', *International Journal of Computer Applications in Technology*, Vol. 67, Nos. 2/3, pp.168–175.

Ghanam, Y. and Maurer, F. (2009) 'Extreme product line engineering: managing variability and traceability via executable specifications', *Agile Conference*, IEEE, pp.41–48. Doi: 10.1109/AGILE.2009.12.

Hinterreiter, D., Prähofer, H., Linsbauer, L., Grünbacher, P., Reisinger, F. and Egyed, A. (2018) 'Feature-oriented evolution of automation software systems in industrial software ecosystems', *Proceedings of the 23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'18)*, IEEE, Vol. 1, pp.107–114. Doi: 10.1109/ETFA.2018.8502557.

Ignaim, K. (2021) '*EvoSPL: An Evolutionary Approach for Adopting Software Product Lines in the Automotive Industry*, PhD Thesis, Escola de Engenharia, Universidade do Minho, Braga, Portugal.

Ignaim, K. and Fernandes, J.M. (2019) 'An industrial case study for adopting software product lines in automotive industry: an evolution-based approach for software product lines (EVOA-SPL)', *Proceedings of the 23rd International Systems and Software Product Line Conference*, Vol. B, pp.183–190. Doi: 10.1145/3307630.3342409.

Lahiani, N. and Bennouar, D. et al. (2017) 'A DSL-based approach to product derivation for software product line', *Acta Informatica Pragensia*, Vol. 5, No. 2, pp.138–143.

Lee, K. and Kang, K.C. (2010) 'Usage context as key driver for feature selection', *International Conference on Software Product Lines*, Springer, pp.32–46.

Maccari, A. and Heie, A. (2005) 'Managing infinite variability in mobile terminal software', *Software: Practice and Experience*, Vol. 35, No. 6, pp.513–537 Lindohf. Doi: 10.1002/spe.645.

Monestel, L., Ziadi, T. and Jézéquel, J-M. (2002) 'Product line engineering: product derivation', *Workshop on Model Driven Architecture and Product Line Engineering*, pp.1–5.

O'Leary, P., McCaffery, F., Thiel, S. and Richardson, I. (2012) 'An agile process model for product derivation in software product line engineering', *Journal of Software: Evolution and Process*, Vol. 24, No. 5, pp.561–571.

O'Leary, P., Richardson, I. and Thiel, S. (2008) 'Developing a product derivation process framework for software product line organisataions', *Conference Proceedings*, Ireland.

O'Leary, P., Richardson, I. and Thiel, S. (2010) 'Improving product derivation in software product line engineering', *Informatik Journal*, pp.65–69.

Pohl, K., Böckle, G. and Van Der Linden, F.J. (2005) 'Software product line engineering: foundations, principles and techniques', *Management for Professionals*, Springer. Doi: 10.1007/3-540-28901-1.

Rabiser, R., Oâ'Leary, P. and Richardson, I. (2011) 'Key activities for product derivation in software product lines', *Journal of Systems and Software*, Vol. 84, No. 2, pp.285–300. Doi: 10.1016/j.jss.2010.09.042.

Sinnema, M., Deelstra, S., Nijhuis, J. and Bosch, J. (2006) 'Modeling dependencies in product families with COVAMOF', *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)*. Doi: 10.1109/ECBS.2006.49.

Steger, M., Tischer, C., Boss, B., Müller, A., Pertler, O., Stolz, W. and Ferber, S. (2004) 'Introducing PLA at Bosch gasoline systems: experiences and practices', *Proceedings of the 3rd International Conference on Software Product Lines (SPLC'04)*, Springer, pp.34–50. Doi: 10.1007/978-3-540-28630-1_3.

Thao, C., Munson, E.V. and Nguyen, T.N. (2008) 'Software configuration management for product derivation in software product families', *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'08)*, IEEE, pp.265–274. Doi: 10.1109/ECBS.2008.53.

Tryggeseth, E., Gulla, B. and Conradi, R. (1995) 'Modelling systems with variability using the PROTEUS configuration language', *Selected papers from the ICSE SCM-4 and SCM-5 Workshops on Software Configuration Management*, pp.216–240.

Van der Linden, F., Schmid, K. and Rommes, E. (2007) 'The product line engineering approach', *Software Product Lines in Action*, Springer, pp.3–20.

Van Gurp, J. and Prehofer, C. (2006) 'Version management tools as a basis for integrating product derivation and software product families', *Proceedings of VaMoS*, Vol. 6, pp.48–58.

Van Ommering, R., Van Der Linden, F., Kramer, J. and Magee, J. (2000) 'The Koala component model for consumer electronics software', *Computer*, Vol. 33, No. 3, pp.78–85. Doi: 10.1109/2.825699.

Note

1 The tool and its documentation are available online at: https://github.com/it-karam/friendlyMapper