Variability Management meets Microservices:
Six Challenges of Re-Engineering Microservice-Based Webshops

Wesley K. G. Assunção
Federal Univ. of Technology - Paraná
Toledo, Brazil
Pontifical Catholic University
of Rio de Janeiro
Rio de Janeiro, Brazil
wesleyk@utfpr.edu.br

Jacob Krüger
University of Toronto
Toronto, Canada
Otto-von-Guericke-University
Magdeburg, Germany
jkrueger@ovgu.de

Willian D. F. Mendonça
Federal University of Paraná
Curitiba, Brazil
williandouglasferrari@gmail.com

ABSTRACT

A microservice implements a small unit of functionality that it provides through a network using lightweight protocols. So, microservices can be combined to fulfill tasks and implement features of a larger software system—resembling a variability mechanism in the context of a software product line (SPL). Microservices and SPLs have similar goals, namely facilitating reuse and customizing, but they are usually employed in different contexts. Any developer who has access to the network can provide a microservice for any task, while SPLs are usually intended to implement features of a single domain. Due to their different concepts, using microservices to implement an SPL or adopting SPL practices (e.g., variability management) for microservices is a challenging cross-area research problem. However, both techniques can complement each other, and thus tackling this problem promises benefits for organizations that employ either technique. In this paper, we reason on the importance of advancing in this direction, and sketch six concrete challenges to initiate research, namely (1) feature identification, (2) variability modeling, (3) variable microservice architectures, (4) interchangeability, (5) deep customization, and (6) re-engineering an SPL. We intend these challenges to serve as a starting point for future research in this cross-area research direction—avoiding that the concepts of one area are reinvented in the other.

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; **Software evolution**; **Software as a service orchestration system**.

KEYWORDS

software product line, microservices, cloud computing, variability management, re-engineering

1 INTRODUCTION

Microservices are small and autonomous services that work together by communicating via lightweight protocols [29]. Usually, microservices are independent from each other, allowing developers to freely choose and combine different technologies regarding, for instance, programming languages, databases or communication protocols [13]. So, microservices are highly interoperable, enabling developers to integrate functionality of different systems that are not implemented with the same technologies [18]. For instance, implementing a complex business rule may require the coordination among a Java, a PHP, and a COBOL application [32, 36]. A microservice-based architecture allows to easily manage and coordinate this combination. Using microservices promises several benefits, such as reduced maintenance effort, increased availability, simplified integration of innovative features, enabled continuous delivery and DevOps, optimized scalability management, as well as reduced time to market [26, 34]. Different organizations, such as Netflix[1] and Uber [14], have successfully adopted this development paradigm to architect their software systems.

Currently, organizations of all sizes have been migrating their legacy systems to a microservice-based architecture to modernize their software systems [8, 11]. The broad adoption and popularization of microservices has caught the attention of the software-engineering research community, particularly because microservices emerged in industry and have only recently been investigated with academic studies.[2] There are several open challenges and gaps of using microservices that are important for practice and research alike [15–17]. One particular research problem is related to managing the variability and fast evolution of microservices. For example, different microservices can be freely combined, can provide the same functionality in different variations, but may be deprecated or offline at any point in time. So, microservice-based systems allow for reuse and customization [7], which are the core principles of software product-line engineering—resulting in similar challenges concerning, for instance, variability management.

A software product line (SPL) allows to systematically reuse software features based on a configurable platform, enabling developers to implement a family of products, which share a common

---

[1]http://techblog.netflix.com/2015/02/a-microscope-on-microservices.html
[2]https://martinfowler.com/articles/microservices.html

base, and customize each product to customer-specific requirements [2, 31, 35]. To this end, the platform comprises configuration options that define what features are variable, usually using the Boolean option of either enabling or disabling a feature. In practice, SPLs have shown to provide several benefits, such as reduced development and maintenance costs, improved software quality or faster time to market [35]. The success of an SPL depends on several factors, such as the coordination among development teams and the technology chosen to manage variability. However, an SPL does usually not focus on improving non-functional properties, such as response time or scalability. So, a traditional SPL may not be ideal for network-based, service-oriented systems in which microservices are employed. Organizations are particularly interested in modernizing their legacy systems by migrating to microservice-based architectures to address such non-functional properties [8].

Microservices' interoperability allows to freely reuse their functionalities. However, managing the resulting variability issues remains a challenging task, as all users share one infrastructure and changing any microservice may affect other users that rely on that microservice. We are aware of few studies that investigate customization in microservice-based systems, but they have not been evaluated in real-world settings [30, 33]. In addition, these studies do not focus on variability management. To tackle this problem, we propose this challenge case, aiming to foster research on applying SPL research and practices to microservices, and the other way around. For this purpose, our contributions in this paper are:

- We propose six challenges that are concerned with managing reuse and variability in microservice-based systems. Precisely, we ask the research community to work on: feature identification and mapping (Section 3.1), variability modeling (Section 3.2), microservice-based product-line architectures (Section 3.3), microservice interchanging (Section 3.4), deep customization of microservices (Section 3.5), and re-engineering a complete SPL (Section 3.6).
- We provide an open-access repository of six open-source webshops that are based on microservices, serving as comparable subjects for our challenge case.[3]

With our challenge case, we aim to initiate cross-area research that solves recent practical problems, and avoids resolving the same problems from scratch. Furthermore, we envision the construction of a body of knowledge on this cross-area research of microservices and variability management.

2 SUBJECT SYSTEMS

For this challenge case, we selected six microservice-based webshops from GitHub. We picked these systems from a curated list[4] to limit the scope of this challenge and provide identical versions as baseline for all solutions, but additional systems from that list may be considered in future work. In Table 1, we provide a summary of the six webshops, including their names, examples for the technologies[5] used (e.g., programming languages and frameworks),

the number of microservices, and the number of commits. We remark that we do not list all technologies completely and in their actual complexity (e.g., databases, some frameworks, and libraries are missing), we provide only an impression of important technical aspects for each webshop. To ensure that all solutions of this challenge use the same versions of these subject systems, improving comparability and replicability, we provide an online repository comprising each system in a separate branch.[3]

The six webshops are:

- **eShopOnContainers** is an online shop that sells various physical products. It is a cross-platform .NET system with sample microservices (i.e., Azure Kubernetes) that runs on Linux, Windows, and macOS using Docker containers.
- **Hipster Shop** is a web-based e-commerce app allowing users to browse items, add them to a cart, and purchase them. Google develops this system and uses it to demonstrate the application of technologies like Kubernetes/GKE, Istio, Stackdriver, gRPC, and OpenCensus.
- **Shopping Cart** is a simple webshop demo application developed using microservices with the .NET Core stack.
- **Sock Shop** is an online shop that sells socks. It is intended to aid the demonstration and testing of microservice and native cloud technologies. The webshop is based on Spring Boot, Go kit, Node.js, and is packaged in Docker containers.
- **Stan's Robot Shop** is a simple e-commerce storefront that includes a product catalogue, user repository, shopping cart, and order pipeline. This application is used as a sandbox to test and learn containerized application orchestration and monitoring techniques.
- **Vert.x Micro-shop** is a complete online-shopping microservice application developed with Vert.x.

In Table 1, we provide links to further information for some of these systems, but we purposely selected webshops with detailed documentations in their respective repository, described by the original authors in a readme file.

3 THE CHALLENGES

In this section, we define six challenges that we observed in current research and practice on microservices, serving as an initial agenda for future research. To this end, we first describe and define each challenge in its general context before *motivating* its importance for the SPL research community, describing the concrete *task* that should be solved, and defining our *evaluation* criteria. Our challenges are closely related to problems tackled in SPL research (e.g., feature identification and location [12, 22], feature modeling [10, 28] or re-engineering [4, 21]) and to the Apo-Games challenge [23], but require that such research is adopted to microservices. We propose to tackle the challenges of (1) identifying the features of microservice-based systems and establishing a mapping to the microservices that implement them; (2) developing variability-modeling techniques that allow to manage commonalities and variability among microservice-based systems; (3) defining a product-line architecture that allows to implement microservices as an SPL; (4) proposing techniques to manage the interchange of microservices of different systems to enable software reuse; (5) customizing microservice implementations on a fine-grained level;

---

[3]https://github.com/jacobkrueger/SPLC2020-Microservices-Challenge.
Will be linked with Zenodo if accepted and DOI assigned to paper
[4]https://github.com/davidetaibi/Microservices_Project_List
[5]A curated list of microservice-related principles and technologies is available at:
https://github.com/mfornos/awesome-microservices

Table 1: Overview of the subject systems for our challenge case.

Name	Technologies (examples)	# Microservices	# Commits
eShopOnContainers [a] GitHub Link:	.Net Core, Docker, Azure, Kubernetes https://github.com/dotnet-architecture/eShopOnContainers	8	3,499
Hipster Shop GitHub Link:	Go, C#, Node.js, Java, Python, Docker, Kubernetes https://github.com/GoogleCloudPlatform/microservices-demo	11	458
Shopping Cart GitHub Link:	C#, .Net Core, RabbitMQ https://github.com/thangchung/ShoppingCartDemo	9	40
Sock Shop [b] GitHub Link:	Go, Node.js, Java, .Net Core, Docker, Spring Boot, Kubernetes https://github.com/microservices-demo/microservices-demo	8	1,612
Stan's Robot Shop [c] GitHub Link:	Node.js, Java, Python, Golang, PHP, RabbitMQ, Docker, Kuberntes https://github.com/instana/robot-shop	7	251
Vert.x Micro-shop [d] GitHub Link:	Java, Vert.X, Docker https://github.com/sczyh30/vertx-blueprint-microservice	8	86

[a] Further details are available at: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/introduce-eshoponcontainers-reference-app

[b] A YouTube video by Luke Marsden shows additional details: https://www.youtube.com/watch?v=zzSElP8pQUA

[c] Further details are available at: https://www.instana.com/blog/stans-robot-shop-sample-microservice-application/

[d] Further details are available at: http://www.sczyh30.com/vertx-blueprint-microservice/

and (6) re-engineering microservice-based systems towards an SPL. While each challenge can be addressed independently, they are related to each other and to the development life-cycle of an SPL, namely analysis, design, implementation, and evolution, and thus promote solving them in combination. For example, a solution may define a variability model and the corresponding product-line architecture, which are two fundamental artifacts to implement an SPL [5]. So, we encourage authors to propose solutions to a subset of our challenges that they consider most interesting.

We intend our challenges to provide the basis for future research, and the same accounts for the challenge solutions. So, for reporting a solution to this challenge case, we ask that all steps, problems, results, analyses, and lessons learned are documented and reported. We ask researchers to make as many details, data, tools, documentation, and other artifacts as possible publicly available, potentially in an open-access repository. This allows for a detailed evaluation of each solution and a comparative analysis between them. Each challenge may be addressed manually (as far as possible) or with (semi-)automated techniques. We do not expect that all subject systems are considered for a solution, but ask for a justification why systems were (not) selected. Finally, we remark that we provide the full development history in each of the branches we extracted, which may be used for more detailed analyses.

3.1 Feature Identification and Mapping

Microservices are designed with the functional scope of micro-tasks. Micro-tasking describes the decomposition of a program task into small and self-contained units of work [1]. So, a micro-task contributes to solving a more complex task with a specific goal in that task's context and environment [25]. As a result, a feature (or functionality, business task) is implemented by composing microservices to form a larger functional unit, which is also known

as "Microservice Compositor"[6] or "API Composition"[7] pattern. So, a first challenge for (re-)engineering microservice-based systems is to identify and map features in a system.

> **Challenge 1:** Identify the features of microservice-based systems, establish a mapping to the microservices that implement them, and compare between systems.

Motivation. Identifying and mapping features is an important activity during domain analysis that helps to reason about the features that should be part of an SPL. This information is important to design the variability model that may be used to manage the SPL. Further, understanding how and based on what microservices each feature is composed helps to design a suitable architecture. Finally, the results of this analysis can help to derive new features that can be composed from already existing microservices. So, the results of such an analysis provide the basis for any further (re-)engineering towards an SPL, and for establishing variability management.

Task. We ask for any solution, for example, based on existing feature-identification techniques or a manual process, that identifies features and establishes a mapping to microservices. By identifying which microservices are involved in a feature, we can enable systematic variability management, avoid duplicated implementations, and understand what variants can be build. To learn from the solution, we ask for a detailed description of the method and results. In particular, we are interested in a critical discussion of the problems that are specific to the six microservice-based systems.

Evaluation. We performed an initial feature identification and mapping for the six subject systems ourselves. While we do not claim that it is a perfect ground-truth, our mapping will serve as a baseline to assess each solution. Moreover, we will evaluate the

[6]https://patterns.arcitura.com/microservice-patterns/design_patterns/microservice_compositor

[7]https://microservices.io/patterns/data/api-composition.html

reporting of the results and problems, for which we expect a detailed description of their microservice-specific properties.

3.2 Variability Modeling

Various variability-modeling techniques have been proposed in research [10], with feature models being the most established one in academia and industry [6, 28]. Such a model defines what features of a configurable platform can be combined in what way. As microservices are intended to solve various tasks in different combinations, are interchangeable, and may not be available or have changed at any point in time, existing variability-modeling techniques may fall short to tackle the complexity of microservice-based systems (see also Section 3.4). So, our second challenge is to investigate how to model the variability of microservices.

> **Challenge 2:** Define a variability model representing the commonalities and variability of a microservice-based SPL.

Motivation. Variability management allows to configure different system variants, reusing common features from the integrated platform and customizing with variable features. Such management requires a variability model to represent how the features of a domain are related to each other. Moreover, a variability model is key for other activities in SPL engineering, such as scoping an SPL and evolving it consistently. So, creating a variability model (usually a de-facto standard feature model) is a key activity in the domain engineering of (re-)engineering an SPL [2, 21, 23, 31].

Task. A solution for this challenge should describe how a variability model for the six webshops has been constructed based on what methods or techniques. The details of this description should at least clarify the input artifacts, the process employed, the constructed model, the challenges faced, and the adaptations to the used modeling technique, process or tool to consider microservices. Also, the derived variability model should be provided in a format that can be inspected with an open-source tool (e.g., FeatureIDE [27]).

Evaluation. For the evaluation, we will consider the details and reasoning provided in the solution's description, particularly how the variability model was constructed and potentially adopted for microservices. As various techniques, for example, static analysis, dynamic analysis, and information retrieval, may be used, we will evaluate to what extent the variability model allows to instantiate the six webshop systems—measuring the precision and recall for that purpose [3]. This is a common evaluation method to reason about the design of a feature model during re-engineering [3, 28].

3.3 Microservice-Based SPL Architecture

Microservices are unique in the sense that they allow to combine various technologies, such as programming languages and frameworks. So, to improve reusability and manage their variability, microservices require a different architecture compared to other software systems. Our third challenge is concerned with this architectural perspective, asking to design an architecture that facilitates the variability management and systematic reuse of microservices.

> **Challenge 3:** Define a product-line architecture that allows to engineer and customize different variants by systematically managing and reusing microservices.

Motivation. A product-line architecture is the core artifact of SPL engineering, defining how the configurable platform is designed and can be configured on implementation level [5]. Having a well-defined architecture based on a variability mechanism is a prerequisite to derive actual variants, maintain the SPL, and evolve it consistently. The architectural perspective can define various levels of detail regarding the SPL, ranging from high-level abstractions to detailed representations of the implementation. However, for representing a microservice-based system, existing techniques for representing and defining a product-line architecture may not be ideal, asking to study and potentially refine these.

Task. We ask the community to derive architectural models for the six webshops that can represent a common platform for all of them. So, the model should consider the various technologies and allow to understand how they interact and may be combined. The solutions may rely on any representation, such as the architecture description language (ADL), unified modeling language (UML), systems modeling language (SysML), or architecture frameworks. However, it should be clearly described what method was used and potentially adapted, how the architecture was derived (i.e., top-down, bottom-up), how the particular challenges of microservices were tackled, and how the final architecture aligns to the webshops.

Evaluation. Identical to the previous challenge, we will evaluate whether the product-line architecture properly describes the six subject systems. In addition, we will review the processes of scoping the modeling technique and deriving the actual architecture. To this end, we ask that the resulting artifacts (i.e., models) can be analyzed with an open-source tool to facilitate their evaluation.

3.4 Microservice Interchanging

Microservices are designed to be well-modularized, with their communication relying on lightweight protocols. These properties leverage the heterogeneous interoperability of microservices. So, interoperability allows to integrate microservices from different systems implemented with different programming languages and platforms [18]. Despite their interoperability, it remains problematic to use microservices of different systems to implement a feature, for instance, due to varying communication interfaces. For example, among our six subject systems, only Hipster Shop comprises a microservice for advertising named adservice. The question is how can we reuse this microservice in another systems? Our fourth challenge is concerned with investigating this problems of interchanging micrsoervices of different legacy systems in an SPL.

> **Challenge 4:** Propose a solution that allows to interchange microservices of different technologies within a system.

Motivation. The microservices of our subject systems rely on different technologies and specific communication protocols. To actually utilize an SPL that allows to derive variants with dedicated features from these microservices, it is necessary to either re-engineer all services to unify their technologies as far as needed or to propose techniques to manage their differences. As for cyber-physical systems [24], we argue that microservices would highly benefit from the second solution, allowing to combine microservices freely, rely on various techniques and providers, as well as facilitating adaptations at runtime (e.g., integrating new services or

replacing deprecated ones). An existing technique for this purpose is called Interchange Context,[8] a technique that provides the same protocol to microservices to communicate. However, tackling this challenge within an SPL arguably requires adaptations to cope with microservices, features, variability, and runtime adaptations.

Task. Tackling this challenge may require to adapt techniques for dynamic SPLs to microservices, or propose a completely new technique that allows to combine the services of our subject systems that rely on different technologies. We ask for a description of how the technique works, whether existing techniques have been adapted, and to provide an evaluation. To this end, a solution should report how the technique has been tested based on what evaluation criteria, and discuss potential shortcomings.

Evaluation. We will consider how a solution solves the challenge of interchanging microservices in an SPL based on the provided descriptions. In particular, we expect that each solution provides a replicable evaluation environment, ideally prepared as a directly usable virtual machine. This environment should, at least, allow to observe the behavior of the proposed technique in practice (i.e., it should interchange microservices at runtime). Moreover, a solution may evaluate the performance of the technique, showing and potentially comparing its scalability. In the end, any solution should measure what microservices can be freely interchanged, whether the interchanges work correctly at all times, and how much overhead (e.g., waiting time) may be caused.

3.5   Deep Customization of Microservices

The previous challenge is concerned with interchanging microservices that build on different technologies, which represents coarse-grained reuse and customization. In contrast, more fine-grained adaptations (e.g., on statement level) may be necessary to tackle feature (or microservice) interactions. While this can be easily achieved in traditional SPLs, microservices pose the problem that any small adaptation of their behavior affects all tenants in their context (i.e., applications in the network that rely on the microservice) and may result in breaking changes. For this reason, we define a challenge that is concerned with managing fine-grained changes and feature interactions (i.e., deep customization) of microservices, even if the microservices may be implemented by independent developers.

> **Challenge 5:** Propose a solution to enable deep customization of microservices in their network-based, multi-tenant context.

Motivation. We are aware of few solutions that aim to tackle the problem of deep customization. For example, Chauvel and Solberg [9] proposed *intrusive* microservices that rely on callback code to intrusively execute queries or commands in the other microservices (i.e., similar to aspect-oriented programming [19]). However, this technique faces severe security issues, as intrusive microservices could be provided by any developer and may not be trusted [20]. Another technique was proposed by Nguyen et al. [30], who provide a proof-of-concept for using specific APIs that allow deep customization. Still, this technique does not seem ideal to cope with a microservice-based SPL, because it only deals with customization, but provides no variability management.

---

[8]https://dzone.com/articles/ddd-interchange-context-and-microservices

Task. To tackle this challenge, we ask for techniques that support feature interactions and fine-grained changes in microservices. For this purpose, any set of microservices of the webshops may be used to show that a proposed technique allows one microservice to adapt another. In particular, this technique has to cope with feature interactions of the subject systems on microservice-level, which arguably requires adaptations to existing solutions.

Evaluation. A solution to this challenge may be based on any combination of systems, features, and microservices from our subject systems. We expect that each solution describes the use-case scenario (i.e., why a feature is interacting and how), the binding time, the impact on other microservice, and the actual technique for handling the interaction. Again we ask that an evaluation environment is provided, implementing the use case and feature interaction as executable instance to observe and evaluate its behavior at runtime. To this end, the main criterion is again the correctness of the implemented interactions.

3.6   Re-Engineering a Microservice-Based SPL

Last, we are concerned with the actual re-engineering of microservice-based systems of one domain towards an SPL. The resulting SPL can enable an organization to systematically manage and reuse its microservices, facilitate the integration of external microservices, and optimize the deployment to customers. Still, we require re-engineering experiences to support organizations in their decision-making on whether and to what extent a microservice-based SPL is useful for them, and to understand practical as well as research problems that need to be solved [21]. While this re-engineering may be based on currently existing SPL concepts, it can also benefit from solutions to any of the previous challenges.

> **Challenge 6:** Re-engineer a microservice-based SPL from systems in the same domain that allows to derive microservice-based products with different configurations.

Motivation. We argue that the advantages of microservices, namely their interoperability and modularity, ease the process of re-engineering different systems into a configurable platform. However, it is unclear to what extent this can be done based on existing SPL re-engineering concepts or requires adaptations to these [4]. A microservice-based SPL can provide additional benefits to an organization, for example, composing features from microservices that may be based on different technologies. For research, it is important to understand not only adaptations to re-engineering processes, but to also assess the benefits and problems of systematic variability management meeting microservices.

Task. We ask to re-engineer the implementation of the six webshops into a microservice-based SPL. The SPL shall allow to configure and derive products at design time or allow for systematic adaptation at runtime. So, the variability mechanism and binding time are up to the concrete solution, but the challenges of re-engineering and deciding on these properties should be part of the description. In particular, existing techniques and tools may require adaptations, as they can potentially not cope with microservices.

Evaluation. To evaluate solutions to this challenge, we require the actual implementation in an evaluation environment (cf. previous two challenges) that allows to configure, derive, and execute

specific products. At least the subject systems that have been re-engineered must be derivable from the SPL, for which precision and recall should be measured. We also expect that the process, efforts, problems, and adaptations of the re-engineering are reported, particularly considering the adaptations required for microservices.

4 SUMMARY

In this paper, we introduced the idea of variability management in the context of microservices. The development paradigms of microservices and SPLs have similar goals, namely facilitating reuse and customizing, but they are investigated separately so far. We presented six challenges that cover the development of an SPL, relating to microservices as the main unit of functionality. Based on these challenges, we intend to combine the benefits of both, variability management and microservices. Consequently, we hope that our challenge case serves as a guide for future research and facilitates the collaboration between the different research communities. In particular, we hope that this research can reveal what techniques of either community require adaptations or can be used as they are.

ACKNOWLEDGMENTS

REFERENCES

[1] Emad Aghayi, Thomas D. LaToza, Paurav Surendra, and Seyedmeysam Abolghasemi. 2019. Implementing Microservices through Microtasks. https://arxiv.org/abs/1903.01977 arXiv.
[2] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2016. *Feature-Oriented Software Product Lines.* Springer.
[3] Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. 2017. Multi-Objective Reverse Engineering of Variability-Safe Feature Models based on Code Dependencies of System Variants. *Empirical Software Engineering* 22, 4 (2017), 1763–1794.
[4] Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. 2017. Reengineering Legacy Applications into Software Product Lines: A Systematic Mapping. *Empirical Software Engineering* 22, 6 (2017), 2972–3016.
[5] Wesley K. G. Assunção, Silvia R. Vergilio, and Roberto E. Lopez-Herrejon. 2020. Automatic Extraction of Product Line Architecture and Feature Models from UML Class Diagram Variants. *Information and Software Technology* 117 (2020).
[6] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS).* ACM, 1–8.
[7] Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rodrigo Bonifácio, Leonardo P. Tizzei, and Thelma E. Colanzi. 2019. Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study. In *International Systems and Software Product Line Conference (SPLC).* ACM, 26–31.
[8] Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rafael de Mello, and Maria J. de Lima. 2019. Analysis of the Criteria Adopted in Industry to Extract Microservices. In *International Workshop on Conducting Empirical Studies in Industry and International Workshop on Software Engineering Research and Industrial Practice (CESSER-IP).* IEEE, 22–29.
[9] Franck Chauvel and Arnor Solberg. 2018. Using Intrusive Microservices to Enable Deep Customization of Multi-Tenant SaaS. In *International Conference on the Quality of Information and Communications Technology (QUATIC).* IEEE, 30–37.
[10] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS).* ACM, 173–182.
[11] Paolo di Francesco, Patricia Lago, and Ivano Malavolta. 2018. Migrating Towards Microservice Architectures: An Industrial Survey. In *International Conference on Software Architecture (ICSA).* IEEE, 29–38.

[12] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature Location in Source Code: A Taxonomy and Survey. *Journal of Software: Evolution and Process* 25, 1 (2013), 53–95.
[13] Nicola Dragoni, Saverio Giallorenzo, Alberto L. Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: Yesterday, Today, and Tomorrow. In *Present and Ulterior Software Engineering.* Springer, 195–216.
[14] Susan J. Fowler. 2016. *Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization.* O'Reilly.
[15] Javad Ghofrani and Daniel Lübke. 2018. Challenges of Microservices Architecture: A Survey on the State of the Practice. In *Central European Workshop on Services and their Composition (ZEUS).* 1–8.
[16] Benjamin Götz, Daniel Schel, Dennis Bauer, Christian Henkel, Peter Einberger, and Thomas Bauernhansl. 2018. Challenges of Production Microservices. *Procedia CIRP* 67 (2018), 167–172.
[17] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The Journey So Far and Challenges Ahead. *IEEE Software* 35, 3 (2018), 24–35.
[18] Muhammad Aslam Jarwar, Sajjad Ali, Muhammad Golam Kibria, Sunil Kumar, and Ilyoung Chong. 2017. Exploiting Interoperable Microservices in Web Objects enabled Internet of Things. In *International Conference on Ubiquitous and Future Networks (ICUFN).* IEEE, 49–54.
[19] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-Oriented Programming. In *European Conference on Object-Oriented Programming (ECOOP).* Springer, 220–242.
[20] Sebastian Krieter, Jacob Krüger, Nico Weichbrodt, Vasily A. Sartakov, Rüdiger Kapitza, and Thomas Leich. 2018. Towards Secure Dynamic Product Lines in the Cloud. In *International Conference on Software Engineering - New Ideas and Emerging Results (ICSE-NIER).* ACM, 5–8.
[21] Jacob Krüger and Thorsten Berger. 2020. Activities and Costs of Re-Engineering Cloned Variants into an Integrated Platform. In *International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS).* ACM, 21:1–10.
[22] Jacob Krüger, Thorsten Berger, and Thomas Leich. 2019. Features and How to Find Them: A Survey of Manual Feature Location. In *Software Engineering for Variability Intensive Systems.* CRC, 153–172.
[23] Jacob Krüger, Wolfram Fenske, Thomas Thüm, Dirk Aporius, Gunter Saake, and Thomas Leich. 2018. Apo-Games: A Case Study for Reverse Engineering Variability from Cloned Java Variants. In *International Systems and Software Product Line Conference (SPLC).* ACM, 251–256.
[24] Jacob Krüger, Sebastian Nielebock, Sebastian Krieter, Christian Diedrich, Thomas Leich, Gunter Saake, Sebastian Zug, and Frank Ortmeier. 2017. Beyond Software Product Lines: Variability Modeling in Cyber-Physical Systems. In *International Systems and Software Product Line Conference (SPLC).* ACM, 237–241.
[25] Thomas D. LaToza and Andre van der Hoek. 2016. Crowdsourcing in Software Engineering: Models, Motivations, and Challenges. *IEEE Software* 33, 1 (2016), 74–80.
[26] Welder Luz, Everton Agilar, Marcos C. de Oliveira, Carlos E. R. de Melo, Gustavo Pinto, and Rodrigo Bonifácio. 2018. An Experience Report on the Adoption of Microservices in Three Brazilian Government Institutions. In *Brazilian Symposium on Software Engineering (SBES).* ACM, 32–41.
[27] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE.* Springer.
[28] Damir Nešić, Jacob Krüger, Ştefan Stănciulescu, and Thorsten Berger. 2019. Principles of Feature Modeling. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE).* ACM, 62–73.
[29] Sam Newman. 2015. *Building Microservices.* O'Reilly.
[30] Phu H. Nguyen, Hui Song, Franck Chauvel, Roy Muller, Seref Boyar, and Erik Levin. 2019. Using Microservices for Non-Intrusive Customization of Multi-Tenant SaaS. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE).* ACM, 905–915.
[31] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques.* Springer.
[32] Mark Richards. 2015. *Microservices vs. Service-Oriented Architecture.* O'Reilly.
[33] Hui Song, Franck Chauvel, and Arnor Solberg. 2018. Deep Customization of Multi-Tenant SaaS Using Intrusive Microservices. In *International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER).* ACM, 97–100.
[34] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2017. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. *IEEE Cloud Computing* 4, 5 (2017), 22–32.
[35] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Springer.
[36] Eric Yuan. 2019. Architecture Interoperability and Repeatability with Microservices: An Industry Perspective. In *International Workshop on Establishing a Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE).* IEEE, 26–33.