Applying Product Line Testing for the Electric Drive System

Rolf Ebert*
Development Electric Drive, BMW Group, Munich, Germany
www.bmw.de

Jahir Jolianis
Development Electric Drive, BMW Group, Munich, Germany
www.bmw.de

Stefan Kriebel
Development Electric Drive, BMW Group, Munich, Germany
www.bmw.de

Matthias Markthaler
Development Electric Drive, BMW Group, Munich, Germany
RWTH Aachen University, Software Engineering, Aachen, Germany
www.bmw.de

Benjamin Pruenster
Development Electric Drive, BMW Group, Munich, Germany
www.bmw.de

Bernhard Rumpe
RWTH Aachen University, Software Engineering, Aachen, Germany
www.se-rwth.de

Karin Samira Salman
Development Electric Drive, BMW Group, Munich, Germany
www.bmw.de

ABSTRACT

The growth in electrification and digitalization of vehicles leads to increasing variability and complexity of automotive systems. This poses new challenges for verification and validation, identified in a Product Line Engineering case study for the electric drive system. To overcome those challenges we developed a Product Line Testing methodology called TIGRE. In this paper, we present the TIGRE methodology. TIGRE comprises the identification and documentation of relevant data for efficient product line testing and the application of this data in the test management of an agile project environment. Furthermore, we present our experiences from the introduction into a large-scale industrial context. Based on our results from the introduction, we conclude that the TIGRE approach reduces the testing effort for automotive product lines significantly and, furthermore, allows us to transfer the results to untested products.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Product Line Engineering, Product Line Testing, Software Product Lines, Automotive Industry

*The authors list is in alphabetical order.

SPLC '19, September 9–13, 2019, Paris, France

ACM ISBN 978-1-4503-7138-4/19/09…$15.00
https://doi.org/10.1145/3336294.3336318

ACM Reference Format:
Rolf Ebert, Jahir Jolianis, Stefan Kriebel, Matthias Markthaler, Benjamin Pruenster, Bernhard Rumpe, and Karin Samira Salman. 2019. Applying Product Line Testing for the Electric Drive System. In *23rd International Systems and Software Product Line Conference - Volume A (SPLC '19), September 9–13, 2019, Paris, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3336294.3336318

1 MOTIVATION

The automotive industry is facing several challenges ranging from the growing complexity of systems and software [13] to a costumers demand for a shorter time-to-market [34]. To overcome those "automated, connected, electrified and service" challenges the BMW Group set up a strategy [3]. One important part of this strategy is the electrification of the drive system. In the past, the development of the electric drive system focused on individual derivates, *e.g.,* the i3 and the i8. This focus changed and by the year 2025, a broad portfolio of approx. 25 derivates will be electrified [2].

The combination of variants of hardware, software, functionalities, time deviations, and other characteristics exponentially increases the scope to be tested. The combination possibilities enable the BMW Group to offer more customized products. At the same time, it challenges the validation and verification of the products. Approaches to deal with these challenges are Product Line Engineering (PLE) and Software PLE, respectively [8].

There is a lack of Software PLE experience reports from the industry [29]. The few industrial Software Product Line Testing (PLT) approaches focus on solving narrow research challenges instead of the whole product line process [28]. Hence, there is a lack of evidence about the transformation of an application-oriented Verification and Validation (V&V) to PLT in an industrial context.

In a case study for the electric drive system we identified industrial V&V challenges and compared them to existing PLE concepts. Based on the investigation's results, we developed a methodology, called Testing IntelliGent, non-Redundant and Efficient (TIGRE). Hence, our contributions are

- an approach to introduce PLT-methodologies into agile development processes
- TIGRE, a methodology for efficient testing of an automotive product line

The particular goal of TIGRE is to reduce redundant testing effort by intelligent planning and reporting of the test execution.

In the following, Section 2 introduces the electric drive system, before Section 3 presents the challenges related to V&V of this system in the industrial context. Section 4 presents TIGRE and Section 5 presents the results of our case study and lessons learned. Afterwards, Section 6 highlights related work. Section 7 discusses TIGRE and Section 8 concludes.

2 THE ELECTRIC DRIVE SYSTEM

To electrify a wide range of vehicles until 2025, the BMW Group developed the fifth generation of the electric drive system [2]. This fifth generation shows the typical characteristics of a product line. The following section introduces the most relevant characteristics.

2.1 Hardware-Components and Communication Busses

The fifth generation of BMW Group's electric drive systems comprises several hardware components. These components communicate over different bus systems. Most prominent are the highly integrated electric drive component, including the electric machine, transmission and power electronics, the battery system and the charger unit with Alternating Current (AC), Direct Current (DC) and DC/DC converter [2]. Each component is available in different variants. Hence, the system can be modified to suit all sorts of different installation spaces and power requirements.

2.2 Software-Functions and Parameters

The current electric drive system provides more than 60 software intensive functionalities, which are distributed over various hardware components. This distribution may vary for different vehicle derivates [1]. Among the main functionalities, *e.g.,* torque, and energy management, plug-in charging of the electric vehicle is one with the highest degree of variability. Reasons are, *e.g.,* country-specific legal requirements, grid voltages, charging standards, derivate specific vehicle architectures and optional equipment. There are different means to realize this functional variability in the software: One possibility is to implement different software variants; more precisely separate pieces of software, which are implemented in parallel. Another possibility is to have only one piece of software but a flexible parameter-set for each functional variant [41].

A future challenge is that proprietary embedded systems become linked to other systems that are more flexible, interactive, networked and seamlessly connected to Cyber-Physical Systems [7]. One of the first functionalities affected by this trend is the charging functionality. The coming into effect of the ISO15118 [36] sets the foundation for Vehicle-to-Grid functions and wireless high-level communication between the vehicle and the charging station [27].

[1] In the automotive industry, the configuration of a vehicle in a vehicle family is referred to as a *derivate* or model. The BMW 7 Series vehicle family, for example, is configured as a limousine derivate, long version derivate, hydrogen drive derivate, and hybrid derivate.

Summing up, each derivate comprises multiple variants, which differ in hardware, software, and the respective functionalities.

2.3 Development Process

In previous generations, the development process of the electric drive system focused on single derivates. For each derivate there existed dedicated responsibilities and artifacts like project plans, requirements, architectures, and test specifications, *e.g.,* test cases, test plans, and test reports. Hence, transforming the system into a product line means the development process has to be changed. The following two prerequisites have already been established and must be taken into account.

Functional- and Requirements-Based testing. Concurrently with the development process, the emerging functionalities are tested to their requirements. The Specification Method for Requirements, Design, and Test (SMArDT) supports the process [10, 11].

Agile Sprints. At the BMW Group, software and system development are based on agile methods, *e.g.,* Scrum and Large-Scale Scrum [18, 37]. According to these agile methods, new hardware and software are developed and integrated into sprints. Each sprint contains a specification, a development, and a testing phase.

3 CHALLENGES IN THE ELECTRIC DRIVE SECTOR

With the fifth generation of the electric drive system, a powerful product line has been established. This product line is the basis for developing derivates.

To handle the product line and its derivates, processes and artifacts have to be introduced, since this cannot be handled efficiently with the traditional application-oriented methods. One suitable methodology field is PLE and Software PLE, respectively. This section presents the main challenges, which we identified in a case study with the support of domain experts at the BMW Group. These challenges need to be addressed by the PLE approach in the automotive industry.

Establishing Domain and Application Engineering (CH01)

To deploy PLE strategies, it is crucial to establish a common understanding of the difference between domain and application engineering. In [33] these terms are defined as follows:

Domain Engineering is the process of PLE in which the commonality and the variability of the product line are defined and realized.

Application Engineering is the process of PLE, in which the applications of the product line are built by reusing domain artifacts and exploiting the product line variability. In the automotive context the applications are derivates or subsystems of derivates. In a subsystem, like the electric drive system, we also call applications products.

Common Platform The common platform contains all elements from which the products can be built.

Traditionally the development of the electric drive system at the BMW Group is application-oriented. This is because in the beginning, when demand for E-Mobility was not as high as today, the BMW Group focused on two vehicle types, the i3 and the i8. Hence, one of the main challenges in establishing domain engineering is to reveal and document the explicit elements of the common platform. This documentation has to include the element dependencies and variabilities. The information has to be extracted from the existing application-specific artifacts and expert knowledge. In addition, the existing processes and artifacts for applications have to be adapted.

Using Synergies of the Product Line for V&V (CH02)

With the increasing number of variants and the combination of variants, the testing effort for the electric drive system expands exponentially. Since the products have a high degree of reuse of the common platform elements, testing each single variant combination is not necessary. There are synergies between the products. The challenge is to reveal them, and by doing so enable intelligent test planning, which leads to an efficient V&V of the electric drive system.

V&V of Product Lines in an agile development process (CH03)

In an agile development process, each sprint needs a suitable test plan for integration testing. The test plan has to consider the necessary test runs for all testing activities during the sprint. The test plans must not be separate for each derivate. Furthermore, the test plan has to ensure that all variants of the derivates are sufficiently tested and there are no gaps or redundancies. The test runs of the test plan must be distributed among the available test platforms in a way that no conflicts arise, *e.g.,* two test runs on the same test platform at the same time. The challenge is to include all necessary information in the test plan, while being prepared for unforeseen events.

Various Testing Objectives. The purpose of testing may differ in each sprint. Depending on the changes in hardware, software and the status of the project, different test goals will be focused, *e.g.,*

- Verification of new functionality
- Verification of new maturity levels with more robustness
- Verification of bug fixes
- Regression testing of unchanged functionality
- Focus on a special variant or derivate, *e.g.,* if it is close to start-of-production or before a specific milestone like homologation of new components, variants or derivates

In each agile sprint, there is a limited testing time. Since the number of variants increases exponentially, it is not feasible to increase the test resources in the same way. Hence, it might not always be possible to execute all available test cases for each variant in every cycle. It is, therefore, necessary to exploit synergies between the products to efficiently use existing resources.

Flexibility of the Test Plan. Even if a test run is perfectly planned, unplanned obstacles can occur during a sprint, *e.g.,* delayed delivery or damage of software, components or test platforms. To use the remaining testing time efficiently, short-term changes in the test plan must be possible at any time.

Reporting the Test Results (CH04)

Another challenge is reporting test results to the common platform and derivate management. Besides the high expectations regarding quality, the electric drive system needs to ensure safety. The management and domain experts demand specific reports to the results of the test execution. These reports have to be relevant to derivate-focused management and to the common platform management. In contrast to the other challenges, CH04 is focused on an automotive specific subject. Automotive-specific, since the product (electric drive system) is always integrated into a specific derivate and additional safety reporting is required for each derivate.

4 INTRODUCTION OF PLE METHODOLOGIES IN AUTOMOTIVE TESTING WITH THE TIGRE APPROACH

In this section, we propose the TIGRE approach to introduce PLE methodologies in the automotive industry and examined the testing process of the electric drive system. TIGRE targets the design of test plans for product lines in a non-redundant and efficient way. Hence, TIGRE targets the challenges of Section 3.

4.1 Phase one aims at the initial identification and documentation of relevant data for PLT.

4.2 Phase two focuses on the usage of this data for testing activities during each agile sprint.

4.1 Identification of the relevant data for PLT

The first phase of TIGRE addresses the establishment of Domain and Application Engineering (*cf.* **CH01**). This phase needs a high initial effort. Yet, once it is established the activities only have to be repeated in case of changes. Figure 1 gives an overview of the necessary artifacts.

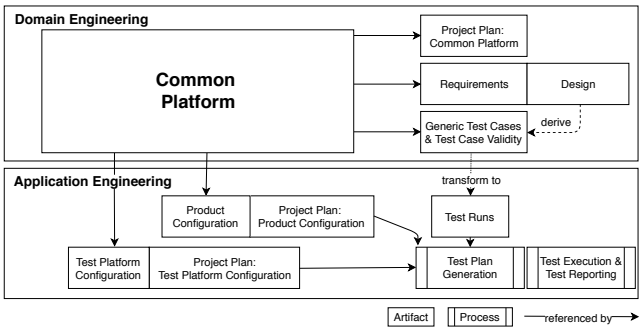

Figure 1: Artifacts and their relations in domain and application engineering

The process of domain engineering comprises the common platform and the corresponding project plan, requirements, design, generic test cases, and test case validity. The process of application engineering comprises the product configurations and test platforms with their respective project plans, and test runs. All those

application engineering artifacts are used for test plan generation in the second phase (*cf.* Section 4.2). The following sections explain each artifact in detail.

4.1.1 The Common Platform. Defining the common platform is the key element for all further activities. The common platform contains all elements from which the products can be built. Each element is described as a variation point according to the Orthogonal Variability Model (OVM) [33]. There are several variants for each variation point. The defined elements of the common platform will subsequently be referred to in all further artifacts, like requirements, design and test specifications, project plans, and for the configuration of the products (see Figure 1).

The high degree of variability in hardware, software, and functionality of the electric drive system results in a large number of variation points. These variation points can be clustered into several groups, which we named *sub-platforms*. As a result, we defined the following sub-platforms:

- A platform of distributed functionalities
- A platform of hardware components
- A platform of software and parameters
- A platform of further characteristics, which give extra information about the electric drive system

Figure 2 illustrates the sub-platforms, including examples for their content. The sub-platform `Functionality` contains one variation point (triangle in Figure 2) for charging functionality `(CHGE)`. `CHGE` has the variants (square) charging with `AC`, charging with `DC` and inductive charging `(IND)`. The sub-platform `Hardware` comprises the variation points high voltage battery `(HV-Battery)`, with the variants `A` and `B`, the electric motor, transmission and power electronics (abbreviation Power Electronics and Electric Machine (PEEM)), with the variants small `S`, medium `M` and large `L`, and the `charging unit (CU)`, with two variants for high power charging `(High)` and low power charging `(Low)`. The sub-platform `Software` contains the variation points Charging Unit Software (`CU-SW`) and the PEEM Software (`PEEM-SW`). Each variation point has two variants, *i.e.,* `SW-A` and `SW-B`. The sub-platform `Characteristics` comprises one variation point, the charging type `(CHGE Type)`. The `CHGE Type` represents four different charging standards. The charging standard `Type-1` is an American standard for AC-charging [19], `Type-2` is a European standard for AC-charging [21, 22], `Combined Charging System (CCS)` is an international standard for combined AC-charging and DC-charging [23], and `CHDMO` is a Japanese standard for DC-charging [20].

4.1.2 Dependencies within the Common Platform. As described in [33] there exist certain dependencies between the elements of the common platform. These dependencies need to be analyzed and documented in order to show valid combinations. Figure 2 illustrates an example of such dependencies: The charging types `Type-1`, `Type-2`, and `CCS` *require* the functionality `AC`. The `AC` functionality *requires* a `CU` and an `HV-Battery` in any variant. The `CU` *requires* a `CU-SW` in any of the available variants.

4.1.3 Requirements, Design and Test Specification. The defined elements of the common platform shall be referred to in the requirements, design and test specifications during domain engineering. For the electric drive system, this is done according to SMArDT: Model-based requirements and design artifacts are elaborated for each functionality of the common platform. Moreover, the artifacts contain information about functional variability. An example of the functional variability is the different behavior of the AC-Charging functionality for the charging standards `Type-1` and `Type-2`.

Based on the SMArDT methodology, generic test cases are generated for each functionality [10, 11]. A generic test case corresponds to a domain test case [33]. Hence, a domain test case can contain variability and should allow efficient reuse in application testing. The set of domain test cases has to cover all functional variants. To realize this, we either define different test cases for each variant or one test case with variable parameters. Since this methodology for test case creation has already been established and published [10, 11], we will not go further into detail. We assume the resulting artifacts are available to the testing activities.

4.1.4 Test Case Validity. The test case validity is an attribute of each generic test case which indicates for which objects the test case is valid and executable.

In the past, due to the application-oriented approach, the test case validity has been documented through an attribute listing for all relevant derivates. With the increasing variability, this is not applicable anymore, because:

- Each derivate may comprise different variants (product configurations) and a test case may not be valid for each variant
- The test case would be executed for each derivate once and there would not be any use of the synergies between the derivates

Therefore, TIGRE comprises a PLE concept of expressing the test case validity based on the defined elements of the common platform. This allows us to use the synergies of the product line as required in **CH02** and reduce testing effort. Instead of performing the test case for each derivate separately, we perform it depending on the common platform elements, which are stated in the validity expression. If the elements are reused by several derivates, the test result is valid for all those derivates. We want to emphasize that this holds for integration testing. Hence, not all components and sub-systems are integrated into the derivate yet. For feature interactions to be tested, this approach refers to the system tests that follow the integration test.

A generic test case can be valid for several variants of elements of the common platform. Hence, two different scenarios for the execution are possible:

- The test case must be performed with several runs, one for each variant it is valid for
- The test case must be performed with only one run for any variant it is valid for and the result can be transferred to the other variants it is valid for

The second scenario allows us to reduce testing effort again and therefore contributes to **CH02**. In both scenarios, each test run can be performed with a different parameterization, depending on the validity of the test case.

For the test case validity, we use information provided in the requirements and design specifications as well as domain experts assessments. The transparency, why a test case needs several runs, is

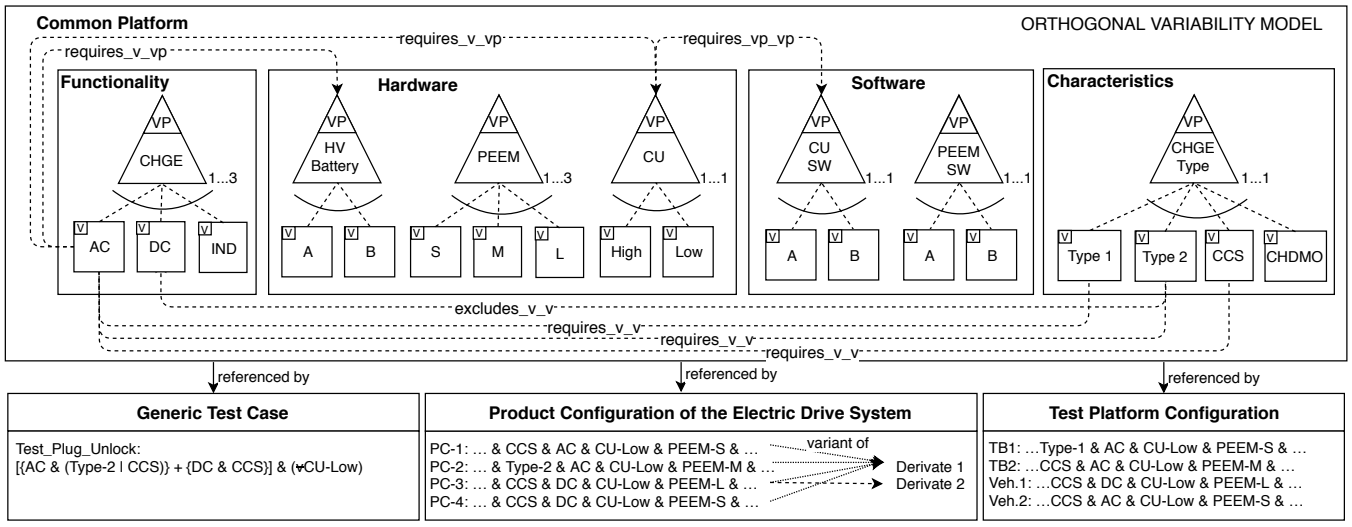

Figure 2: Example of artifacts of the common platform, generic test case, product configuration and test instace configurations

increased compared to a derivate relationship. The test case validity is stored as an attribute of the generic test case. We can change and improve it throughout the development process of the product line. This allows us to take advantage of lessons learned and document the expert knowledge in a reasonable way (*cf.* **CH01**).

To use the test case validity for the planning of the test execution and reporting, it is documented in a specific notation. To express the explained scenarios for test case validity, a notation with logical expressions is feasible. In addition to being machine-readable, this logical expression must also be readable by domain experts, test experts and managers. Therefore, the syntax should be minimalistic in order to facilitate readability and interpretation.

In particular, we identified the need to expand the test case validity expressions by equivalence classes. An equivalence class is a set of variants which a generic test case is valid for. This specific set states that we need only one run of the test with any representative of the equivalence class. The result of this single run can be transferred to all other members of the equivalence class.

In the following, we explain the Operator-Name, the Operator-Symbol, the Operator-Description and the Operator-Example used in our specific notation.

- **Symbol:** |
  Name: logical OR
  Example: A | B
  Description: The test case is valid for A and B, it must be executed once for a product containing either A or B and the result can be transferred to all other products containing A or B. This operator is used to express equivalence classes.
- **Symbol:** &
  Name: logical AND
  Example: A & B
  Description: The test case is valid for A and B, it must be executed once with a product containing A and B.
- **Symbol:** {...}+{...}
  Name: Separate-Runs-Operator
  Example: {A} + {B}
  Description: The test case is valid for A and B, it must be executed twice, once with a product containing A and once with a product containing B.
- **Symbol:** ∀
  Name: All-Operator
  Example: ∀Variation-Point-X
  Description: The test case is valid for all variants of Variation-Point-X, it must be executed separately for each variant.

As an example, we use a generic test case concerning the unlock functionality of the charging plug. The following expression shows its validity attribute (*cf.* Figure 2) *i.e.,* Test_Plug_Unlock:

```
[{AC & (Type-2 |CCS)} + {DC & CCS}] & (∀CU-Low)
```

With this validity attribute, we conclude that we need two separate runs of the test case Test_Plug_Unlock. One test run with a product containing the variants `AC`, and either Type-2 or `CCS` and another run with a product containing the variants `DC` and `CCS`. The expression `(Type-2 | CCS)` forms an equivalence class. Both runs have to be executed with a `CU-Low` variant. The all-operator states that all versions of `CU-Low` have to be tested separately. This will be explained in detail in Section 4.1.7.

4.1.5 Product Configurations. The specific products are managed within the application engineering. A major principle of PLE is that the products are composed of a combination of the reusable elements of the common platform. The specification of the variation points, including their dependencies, enables the generation for possible product configurations.

To set up a complete product configuration, we consider all variation points of the common platform and decide whether they are present or not. If a variation point is present we select at least one of the possible variants. This is in accordance with the defined dependencies and restrictions, as described in Section 4.1.2.

The resulting product configurations are stored in a database. Each configuration is mapped to the derivate where it is present. Figure 2 shows four examples for parts of test-relevant Product Configurations (PC) and their mapping to product, *e.g.,*

$PC-2:$ `...& Type-2 & AC & CU-Low & PEEM-M &...`

In the context of product configuration (*cf. PC-2*), it is a list of the elements that make up the product. Similar to the test platform configuration the elements are connected with the symbol `&`.

4.1.6 Test Platform Configurations. Similar to the product configurations, the test platform configurations are expressed by the common platform elements. A test platform is a test bench or vehicle where a test run can be executed. This expression of the test platform configuration allows easy matching of test case validity, product configurations, and test platforms when planning the test execution. Examples for test platform configurations are depicted in Figure 2. Test Bench 1 has the following configuration:

$TB1:$ `...& Type-1 & AC & CU-Low & PEEM-S &...`

4.1.7 Project Plans. As a prerequisite for planning the test execution for agile sprints, not only *variability in space* but also *variability in time* has to be considered. The variability in time is the existence of different versions of an element of the common platform that are valid at different times [33]. The variability in space is the existence of an element in different shapes at the same time. Therefore, for each variant of the common platform, we have to define whether it is available and in which version, for each specific sprint. An example is depicted in Figure 3. The example shows the different versions of the hardware component variants `CU-Low` and `PEEM-S` during four successive sprints. Moreover, it illustrates, in which sprints the functionality `AC` is available. In sprint 1, both component variants are available in version 1 (`v1`), but `AC` is not present. In sprint 2, `CU-Low` is available in `v2`, while `PEEM-S` is still available with `v1` only. In this sprint, the `AC`-functionality is present for the first time. From now on it will be available in all further sprints. In sprint 3, `CU-Low` is available in `v3` and `PEEM-S` in `v2`. In sprint 4 `CU-Low` is available in `v4` and `PEEM-S` in `v3`. We call this the *project plan of the common platform.*

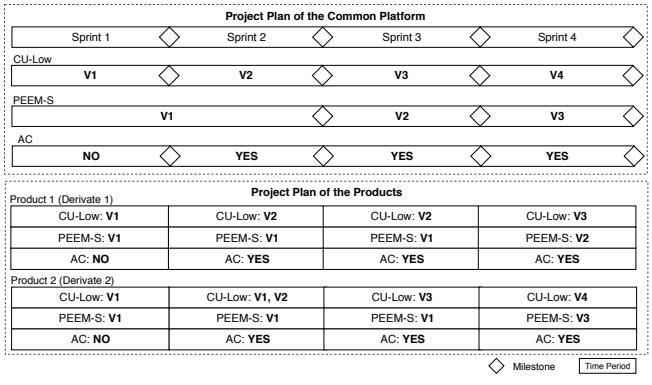

Figure 3: Simplified example of a sprint schedule

To define, which version of a variant is integrated into a product for each agile sprint, a *project plan for products* is necessary as well. Figure 3 illustrates, which versions of `CU-Low` and `PEEM-S` have to be tested for *product 1* and *product 2* during the four sprints. It also shows which products include the AC-functionality and in which sprint it will start to be present. Based on project decisions the following scenarios for integrating available versions from the common platform to products are possible:

- A newly available version in the common platform might be integrated at a later sprint for a special product. For example, in sprint 3 there is a `v2` available for the `PEEM-S`. However, none of the product integrate it in this sprint.
- For one product, there might be two valid versions of the same variant at the same sprint. For example in sprint 2, product 1 uses both versions `v1` and `v2` of the `CU-Low`.

Thus, it is possible that different versions of one product configuration have to be tested separately during one sprint. As an example, we take the product configurations, which have been defined in the previous section (*cf.* Figure 2). If we want to test them in sprint 2, we have to consider the relevant versions defined for sprint 2 in the project plans (*cf.* Figure 3). This leads to the following *sprint-relevant product configurations*:

PC-1: `CCS & AC & CU-Low-v1 & PEEM-S-v1 &...`
PC-2: `Type-2 & AC & CU-Low-v1 & PEEM-M-v1 &...`
PC-3: `CCS & DC & CU-Low-v1 & PEEM-L-v1 &...`
PC-4: `CCS & DC & CU-Low-v2 & PEEM-L-v1 &...`
PC-5: `CCS & DC & CU-Low-v1 & PEEM-S-v1 &...`

These sprint-relevant product configurations have to be considered during the planning of the test execution in the second phase of this approach.

There is also a project plan of the test platforms, which reveals the versions that will be available on the test platform at a certain sprint. From this project plan, the sprint-relevant test platform configurations can be determined with the same method.

4.2 Testing the Product Line in Agile Sprints

The second phase of TIGRE focuses on the planning of the test execution and the reporting of test results. It addresses the challenges **CH02**, **CH03**, and **CH04**.

Figure 4 gives an overview of this phase. In the beginning, relevant information of the before established artifacts is gathered. The data of each artifact are combined to generate a test plan for the test execution. The test plan has to comprise (1) the necessary runs of a generic test case, (2) the product configuration each run has to be performed with, and (3) the test platform each run has to be performed on.

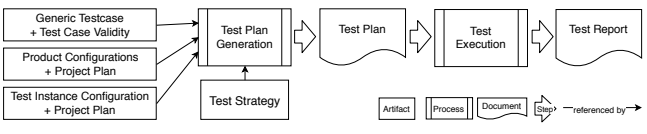

Figure 4: Overview of the interconnected artifacts for the test plan generation

To optimize the test plan, different test strategies can be applied to the generation process. In each agile sprint, a suitable test strategy is selected to fulfill a specific testing objective. After the generation of the test plan, the tests are executed and the test results are reported. In the following sections, we provide a detailed explanation of the *test plan generation* and the *reporting of test results*.

4.2.1 Generating the Test Plan. The generation of the test plan can be divided into three steps.

Step 1. In step 1, the test runs that are relevant for a specific sprint have to be determined. To do this, we filter the project plan for all functionalities that are present in the specific sprint. Based on these functionalities, we choose the related test cases.

For each generic test case, we evaluate the validity attribute to identify which variants need separate runs of the test case. This results in a number of possible test runs. In this way, we transfer the generic domain test case to several test runs for application testing. This step corresponds to the binding of variability [28].

Figure 5 illustrates the calculated test runs for the validity attribute of the example generic test case presented in Figure 2. For the calculation, we also considered the available variant versions in sprint 2 (*cf.* Figure 3). Figure 5 depicts four separate test runs derived from the test case `Test_Plug_Unlock` (*cf.* Figure 2):

Run-1: `AC & (Type-2 | CCS) & CU-Low-v1`
Run-2: `AC & (Type-2 | CCS) & CU-Low-v2`
Run-3: `DC & CCS & CU-Low-v1`
Run-4: `DC & CCS & CU-Low-v2`

The first two test runs contain an equivalence class. To plan the test run for execution, we have to select one representative of the equivalence class. The decision for a suitable representative must be supported by the test strategy. Based on our analyses of frequent testing objectives during an agile sprint, the following test strategies meet our needs:

Random: For each equivalence class, we select a random representative.

Exclusion We define certain variants that should not be selected as a representative for an equivalence class.

Preference We define one or more specific variants that are preferably selected as a representative for an equivalence class.

History-Based For each equivalence class, we select a representative which has not been selected in the past.

Maximal or Minimal Variation For all equivalence classes, we select the representatives so that we receive the minimal or maximal possible number of different representatives.

Combinatorial Interaction Testing (CIT) CIT [12] enables the selection of a subset of products, which most likely reveal the faults caused by interaction. The representatives are selected so that relevant combinations of variants occur, but not all possible combinations. Hence, the testing effort can be reduced.

For our example, we select a history-based strategy and assume that `CCS` has been executed last time, so now we select `Type-2`. This means that the first test run will be executed for a product containing `AC`, `Type-2`, and `CU-Low-v1`. The result will be transferred to all products, which include these variants and also to all products with `CCS`, `AC`, and `CU-Low-v1`.

As shown in Figure 5, each test run inherits a part of the logical validity expression from its parent generic test case. Since this validity expression consists of elements of the common platform, it also represents a part of a possible product configuration. Thus, we call the test runs validity expression a *sub-configuration*.

Step 2. In step 2, we extract the *sprint-relevant product configurations* as described in Section 4.1.7. In our example, the resulting configurations are depicted in Figure 5. Subsequently, we assign the sub-configurations of the test runs to the *sprint-relevant product configurations* (see (A) in Figure 5). This can lead to different results:

- If there is no product configuration containing a certain sub-configuration, the test run must be deleted. In our example, this is the case for the test run 2.
- If there is exactly one product configuration containing a certain sub-configuration, the test run must be executed with this configuration. This is the case for the test run 4.
- If there is more than one product configuration containing a certain sub-configuration, the test run can be performed on any product configuration. The test result can be transferred to the other product configurations. This refers to **CH02**. In the example, this holds for the test run 3.

In the last case, we must select one product configuration for the execution. To support a reasonable decision-making, the test strategies defined in step 1 can be used again.

In the example, we use a test strategy that selects the `PEEM-L` variant with *preference*. As a result, product configuration 3 will be assigned to test run 3.

Step 3. In step 3, we determine the sprint relevant test platform configurations as described in section 4.1.7 and allocate the test runs to test platforms. To do this, we compare the selected product-configurations of each test run with the *sprint-relevant test platform configuration* (see (B) in Figure 5). If a test platform with the required configuration is available, the triple (test run – product configuration – test platform) can be assigned to the test plan. If no test platform is available, we consider the following alternatives:

- If the test run contains an equivalence class, we could take another representative
- If a test run's sub-configuration is present in several product configurations, we can select another product configuration
- If none of the above alternatives led to a result, we have to show the gap in the test report

For the first test run of the example in Figure 5, we recognize that there is no test platform with a sub-configuration `AC & Type-2`. Therefore, we select the other representative of the equivalence class and plan the run `AC & CCS` at test bench 2 (`TB2`).

Figure 5 shows the resulting test plan. It contains all test runs that need to be executed. For each test run, the product configuration and the test platform for the test execution are provided.

4.2.2 Reporting the Test Results. After the execution of the planned test runs, the test results are reported. Hence, we generate dedicated test reports for the common platform and the commercially

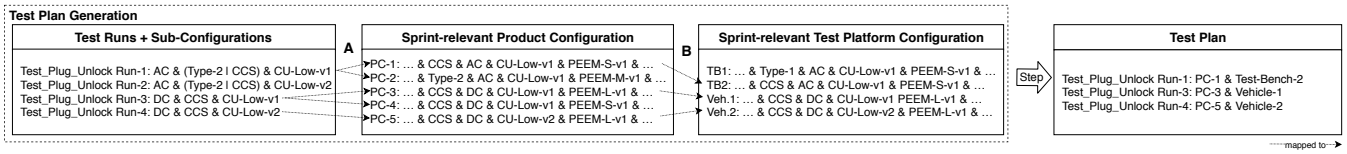

Figure 5: Example of the test plan generation, for a single generic test case, with consideration of the test strategy

relevant products and derivates, respectively (*cf.* **CH04**). These reports provide different views on the test results. Hence, different Stakeholders in the Company get a precise overview of the test results, edited/prepared for their specific view. Those views are automatically derived from the existing test results. The first report lists each test run result, showing the sub-configuration, it was executed with and the configurations where the result can be transferred. The product specific report lists only those test run results that are valid for a sub-configuration of this products. For each test run, we report whether it was executed with this product or transferred from the execution with another product.

5 CASE STUDY AND LESSONS LEARNED FROM EMBEDDING PLT INTO THE AUTOMOTIVE INDUSTRY

After defining the main process and input artefacts, the proposed approach was applied during a pilot phase. The aim of the pilot project was to show the applicability in an industrial environment. Furthermore, the pilot phase already involved members of various teams and helped to spread the basic knowledge about PLE fundamentals and the methodologies.

In the beginning, the relevant data for PLT was elaborated as described in Section 4.1 of the TIGRE approach. Since there was no specific tool support, the complexity for the pilot phase was kept as low as possible. For an integration phase of the electric drive system seven representative drive system functionalities were chosen. For each of those functionalities all the linked test cases were imported from application lifecycle management tools. Hence, we received 163 generic test cases in total. We chose three derivates for which the pilot phase should provide test results. Based on the defined elements of the common platform we derived the sprint relevant product configurations. Although the amount of variation points in the characteristics sub-platform was low, we identified 30 sprint relevant product configurations for the three derivates.

5.1 Pilot phase

The goal of the pilot phase was the manual creation of a test plan for the chosen integration phase. In order to generate the test plan, the following artifacts were created:

- The sprint-relevant product configurations, based on interviews with experts and the information contained in several databases.
- The generic test cases with their validity attributes.
- The sprint-relevant test platforms with their configurations and availability.

After collecting all of the necessary data, exemplary test sets were created following the proposed approach in section Section 4. The result of the pilot phase has been a significant reduction of test runs, compared to just multiplying all of the test cases with the number of sprint-relevant product configurations.

Executing each test case once for each product configuration would result in the multiplication of 163 test cases with the 30 product configurations. However, not all of those 4.890 runs would be reasonable to be executed, since there exist synergies.

Applying the TIGRE methodology, we could reduce the number of necessary test runs to a total of 326 runs, which covers all of the sprint-relevant product configurations for the electric drive system. Furthermore, we were able to trace our results back to all variants and expose those which had not been explicitly tested.

5.2 Lessons learned from applying the methodology

Creating a test plan with a generally defined methodology aims to improve the traceability, reproducibility and optimization of the test plans over time. However, collecting all of the relevant information and combining them was a challenging task.

The generation of the three above mentioned artifacts required access to various tools and expert knowledge as well. The agile development requires agile tools for data maintenance. Project decisions obtained during meetings are traditionally documented in various independent tools and transfered to the actual database in a second step. In this transfer process information could get falsified or even lost. Moreover, specialized tools are often only accessible and understood by experts. Hence, we collaborated closely with those experts. The effort was worth it, since we stored the expert knowledge in tools. Now, all of the expertise can be reused easily and even if there is a personnel change.

Though others see a big challenge in the acceptance of PLE with its development structure [9], TIGRE was welcomed with interest by all involved domain experts, testing experts and managers.

Because of the high complexity of the electric drive system, manually generating a test plan was not feasible with reasonable effort. Hence, the implementation of such a methodology with modern tools is necessary.

6 RELATED WORK

In the past twenty years, several literature reviews have been conducted on PLT, analyzing a number of research approaches in this field. However, in the research field of PLE, the number of studies facing the scale testing of Product Lines in the industry is low [14, 28, 29].

The Software PLE investigation [29] reviewed 276 studies with the scope of identifying test strategies, which facilitate fault detection and reduce the effort for quality assurance. The study considered studies between 1998 and 2013. Only 6% refer to industrial practices and 8% are industrial studies. The investigation concludes that Software PLE studies can be categorized in "selection of products" to test and "actual test of products". However, only in the year 2013 there were studies found taking into consideration both processes. CIT (*cf.* Section 4) is also seen as a principle to reduce the testing effort and targets the challenges of **CH02** and **CH03**. The fundamental assumption of CIT is that most of the errors are caused by a single value or the interaction of two input values [12]. The study [29] identified CIT as the most common approach to reduce testing effort through the selection of products. Hence, TIGRE also comprises this optimization technique.

Apart from one study validated in an industrial context, all other approaches were evaluated in research groups. In the field of test case generation, test case selection and test inputs [29] could not find any available tools. Moreover, the literature review provided no information about studies concerning the test management, test plan generation or test reports.

In [14] a systematic mapping study on Software PLT is presented. Test organization, test processes, and test management are pointed out as relevant fields. The majority of the studies provide proposals for the established challenges in PLT. Nevertheless, only 17% of the studies present the usage and evaluation of the proposed methods. Hence, most of the proposed methodologies assume the existence of idealistic inputs and state clear requirements for the approaches, *i.e.,* complete variability models and properly linked information in the overall development process. In an industrial environment, those methodologies have to prove that they work under industrial circumstances. Those applicable test methodologies and strategies are hard to find [14].

Another survey [28] compared and analyzed the contributions of eight software product line specific approaches. The survey provides a framework of relevant artifacts for the overall Software PLT process. The framework enables categorizing and analyzing existing researches, as well as the identification of open research opportunities. The most relevant revealed research opportunities for our research are:

- Criteria for test case selection to reflect the decision on variability resolution and application-specific variability
- Coverage mechanism to determine test case creation, test selection, and execution approaches.
- Reusing or/and modifying test assets for application-specific testing

The framework neither details the artifacts nor explains the artifact's interaction for direct and profitable industry application. Agile methods are not considered. The conclusion of [28] is that most of the approaches focus on narrow Software PLT research challenges. Hence, the approaches do not provide a holistic software product line process from the beginning until completion. Furthermore, the survey stated that only marginal research on how to reduce redundant testing effort by PLT techniques exist. This reduction of redundant testing effort by PLT is one of the most important challenges we address in this paper.

Besides the aforementioned literature reviews, several PLT studies address parts of challenges mentioned in Section 3. The methodology and tooling described in [35] are based on feature models and focus on Software PLE. The extensive approach tackles the challenges **CH1**, **CH2**, and **CH4**. It distinguishes between the problem domain and the solution domain. The problem domain, which is represented in feature models, expresses similarities and variabilities. Hence, the stakeholders' requirements emerge from this domain. The solution domain, a hierarchical structure, consisting of elements of the architecture, satisfy those requirements. Thus, this hierarchical structure comprises the variation points and the connection of the solution and problem space. The language resulting from the approach expresses constraints, restrictions, and moreover calculations. Similar to our specific notation the language uses operators, *e.g.,* `REQUIRES`, `IMPLIES`, and `CONFLICTS`, in combination with logical operators, *e.g.,* `AND` and `OR`, but with a different interpretation, not in context for testing. The approach focuses on software, hence its testing framework is mainly designed for software tests. As such, we did not find applicable solutions for our specific agile test plan generation. Still, the tool provides a well-developed language and its infrastructure provides a large set of extensions and interfaces.

The study [15] introduces a product line application in the automotive industry and presents approaches to meet the challenges of the "mega-scale" product line. The challenges **CH01** and **CH03** are also identified and necessary measures are presented. Mentioned challenges are product complexity, a high number of variants in automotive applications, feature interactions, the size of the companies, the traceability, and the consistency throughout the life cycle. We have faced those challenges in our study as well. To provide solutions to the architectural decomposition, involved roles in the development and the necessary organizational adoptions are discussed. Despite an overview of the PLE development process, the study does not cover details concerning the test process *i.e.,* test management and test reporting. The subsequent study [41] enhances the predefined "mega-scale" product line challenges with challenges addressing **CH02**. Furthermore, the approach takes advantage of system modeling and feature models. However, no applicable solutions for Software PLT, *i.e.,* the test plan generation and reporting of test results (**CH04**) are presented.

In [17] the challenges of testing the weapon system using PLE approaches and a commercial PLE tool are presented. The study refers to **CH01**, **CH02**, and **CH03**. In order to start testing activities so-called 'branching' is necessary. Thus, a copy of the actual work is done and this release is used as a stable basis for testing activities. If necessary changes to the copy must be done, a strong discipline regarding the changes on the copies is required. Event branches are also created, as during the development of the product line also products must be generated at specific points in time. Although the approach tackles testing under time pressure, the variation of the environment under consideration is comparatively low to the automotive industry. Hence, the presented "copy-approach" is not sufficient for the complexity of the electric drive system. Furthermore, besides capturing the results in a database, the approach does not detail the reporting of the results.

Another approach [6] focuses on **CH01** and presents the possibilities of modeling the problem and solution space with tool support.

The approach shows the design of the variable architecture and product derivation. Testing the product line, however, is mentioned as an open issue of Software PLE. The reduction of the testing effort is addressed as an issue concerning many product lines.

The approach in [30] targets the challenges of **CH02** by reducing the testing time of cyber-physical-systems. In the first step, the approach reduces the testing time by selecting and prioritizing products to be tested. In the next step, multiple testing iterations are performed. In each iteration, a small number of test cases for each product is allocated and the test results feedback is taken into account for the next products and test cases. The approach focuses on reducing testing time, while test platforms and information about the product project management are not considered. The test platforms and the knowledge about availability and deadlines for each drive system variant are essential for our approach. Furthermore, [30] does not consider the aspect of an agile development process (**CH03**) or reporting test results (**CH04**).

The model-based approach in [38] aims to avoid redundant testing in application engineering (**CH02**). In order to avoid the execution of redundant test cases, the approach introduces a dataflow-based coverage criterion. The approach combines the usage of feature models and activity diagrams as test models to link data accordingly. Dependencies between variants are registered using a dependency matrix. The conclusion of [38] is that there is a high potential for increasing test efficiency. However, there is a strong dependency on the implementation of the test cases. Test cases can cover more branches and more than one dependency. Thus this dependency influences the efficiency of the approach. In the context of the electric drive system, the technique of [38] is very interesting for the generation of test cases from system models. However, this only covers functional variability. Variability of hardware, software and time are not considered.

Another approach [32] uses pairwise testing to avoid testing every possible combination of input values (**CH02**). The approach only tests the combination of all pairs. Furthermore, this testing technique could be transferred from input values to the features in a product line. The approach uses CIT based on a feature model of the domain. Based on the dependencies of the features the approach can achieve a reduction of the test cases. Nevertheless, the approach does not address our challenges regarding agile process development and reporting test results.

7 DISCUSSION

Our experiences and conclusions are based on an industrial pilot project. To introduce TIGRE in the electric drive system, we carried out the work to a large extent manually. Although we only considered the electric drive system, we still consider the results to be meaningful because our approach involves relevant domains for many automotive functionalities, *i.e.,* mechanical, electrical, and software domain.

We are aware that feature interactions can be a risk of transferring test results [1, 4, 25, 26, 39, 40]. Nevertheless, it will not always be possible to perform every test with every product configuration, because of restricted time and test resources [39]. Therefore, we have to select the test runs carefully. By documenting the test case validity, TIGRE supports a transparent and reproducible selection of test runs. Moreover, due to the high safety requirements in the automotive domain, possible interactions must be identified before implementation [24]. Hence, in contrast to the telecommunication domain, fewer errors are found due to feature tests that were not already found by functional tests [16]. For feature interactions we recommend methods in development and specific test cases that TIGRE can integrate in system testing [1, 5, 25, 31].

We consider CIT for TIGRE, but because of time constraints, we could not apply the method for the pilot phase. CIT could reduce our testing effort even further.

In the pilot phase, we derived the information about the validity expressions manually out of the SMArDT models and by expert assessments. Despite the high manual effort and the uncertainty about the correctness of the expert knowledge, the approach lead to a higher transparency for test case selection. The decisions can be replicated, discussed and changed by lessons learned.

All the information of existing derivates is stored in a database. Hence, we did not set up feature models or OVMs for the pilot phase. Although OVMs and feature models also contain non-existing configurations, we only need real configurations for our test runs. Our specific notation is simple and exactly adapted to the needs of testing the electric drive system. Due to our specific notation, we could not use an elaborated existing tool for our approach. Depending on the type of tool we use, we may transform our notation. Hence, our generated test plan with its runs only exists in one table form and has to be transferred to the test management tool in a second step. The generation of an executable test plan to the defined test strategy still has to be implemented.

8 CONCLUSION

The increasing number of electrified derivates over the next few years is urging the BMW Group to develop and apply new methodologies. Since the new generation of the electric drive system shows significant product line characteristics, one approach to maintain high-quality standards is the use of PLE methodologies.

TIGRE introduces PLE methodologies to the integration testing of the electric drive system. It comprises several artifacts that need to be established in domain and application engineering. These artifacts facilitate the planning of an efficient test execution with reduced test effort. Thus, we demonstrate a procedure to optimize the assignment of domain test cases, product configurations and test platforms.

In a case study with seven representative functionalities of the electric drive system, we reduced the redundant testing effort from 4.890 test runs to 326 test runs for one agile sprint phase. Moreover, we are able to transfer the test results from tested product configurations to other non-tested product configurations. Hence, the TIGRE methodology allows to take advantage of product line synergies and reduces test effort. It is capable of increasing the efficiency, traceability and transparency for the integration test of the electric drive system.

ACKNOWLEDGMENTS

We thank all participating experts who have supported our work, shared their knowledge and evaluated the TIGRE methodology.

REFERENCES

[1] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 33rd ACMIEEE International Conference on Automated Software Engineering*, Marianne Huchard (Ed.). ACM, [S.l.], 143–154.

[2] Matzner Angelika. 28.12.2018. Interview: Stefan Juraschek, Vice President Development Electric-Powertrain. https://www.press.bmwgroup.com/global/article/detail/T0288899EN/interview:-stefan-juraschek-vice-president-development-electric-powertrain?language=en

[3] Simon Anika. 07.09.2017. Statement Harald Krüger, Chairman of the Board of Management of BMW AG, IAA Preview 2017: Press Release. https://www.press.bmwgroup.com/global/article/

[4] Sven Apel, Joanne M. Atlee, Luciano Baresi, and Pamela Zave. 2014. Feature Interactions: The Next Generation (Dagstuhl Seminar 14281). *Dagstuhl Seminar 14281* (2014).

[5] Sebastian Benz. 2010. *Generating Tests for Feature Interaction.* Dissertation. Technische Universität München, München.

[6] Danilo Beuche and Mark Dalgarno. 2007. Software product line engineering with feature models. *Overload Journal* 78 (2007), 5–8.

[7] Manfred Broy, María Victoria Cengarle, and Eva Geisberger. 2012. Cyber-Physical Systems: Imminent Challenges. In *Large-scale complex IT systems*, Radu Calinescu and David Garlan (Eds.). Lecture Notes in Computer Science, Vol. 7539. Springer, Berlin, 1–28.

[8] Paul Clements and Linda Northrop. 2002. *Software product lines: practices and patterns.* Vol. 3. Addison-Wesley Reading.

[9] Paul C Clements. 2015. Product Line Engineering Comes to the Industrial Mainstream. In *INCOSE International Symposium*, Vol. 25. Wiley Online Library, 1305–1319.

[10] Imke Drave, Timo Greifenberg, Steffen Hillemacher, Stefan Kriebel, Evgeny Kusmenko, Matthias Markthaler, Philipp Orth, Karin Samira Salman, Johannes Richenhagen, Bernhard Rumpe, Christoph Schulze, Michael Wenckstern, and Andreas Wortmann. 2019. SMArDT modeling for automotive software testing. *Software: Practice and Experience* 0, 49 (2019), 301–328.

[11] Imke Drave, Timo Greifenberg, Steffen Hillemacher, Stefan Kriebel, Matthias Markthaler, Bernhard Rumpe, and Andreas Wortmann. 2018. Model-Based Testing of Software-Based System Functions. In *44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 146–153.

[12] Irwin S Dunietz, Willa K Ehrlich, BD Szablak, Colin L Mallows, and Anthony Iannino. 1997. Applying design of experiments to software testing: experience report. In *Proceedings of the 19th international conference on Software engineering*. ACM, 205–215.

[13] Christof Ebert and John Favaro. 2017. Automotive Software. *IEEE Software* 34, 3 (2017), 33–39.

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

[15] Rick Flores, Charles Krueger, and Paul Clements. 2012. Mega-scale product line engineering at general motors. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*. ACM, 259–268.

[16] Brady J. Garvin and Myra B. Cohen (Eds.). 2011. *Feature Interaction Faults Revisited: An Exploratory Study: 2011 IEEE 22nd International Symposium on Software Reliability Engineering.*

[17] Susan P. Gregg, Denise M. Albert, and Paul Clements. 2017. Product Line Engineering on the Right Side of the V. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*. ACM, 165–174.

[18] Mathias Heerwagen. 2018. Entwicklung im Wandel Agile Methoden auf dem Vormarsch. *ATZ - Automobiltechnische Zeitschrift* 120, 4 (2018), 10–15.

[19] Hybrid - EV Committee. 2017. SAE Electric Vehicle and Plug in Hybrid Electric Vehicle Conductive Charge Coupler: SAE J1772.

[20] IEEE. 2016. Std 2030.1.1-2015: IEEE Standard Technical Specifications of a DC Quick Charger for Use with Electric Vehicles.

[21] International Electrotechnical Commission. 2011. Plugs, socket-outlets, vehicle connectors and vehicle inlets - Conductive charging of electric vehicles: IEC 62196.

[22] International Electrotechnical Commission. 2017. Electric vehicle conductive charging system: IEC 61851.

[23] International Electrotechnical Commission. 2018. Road vehicles – Vehicle to grid communication interface: ISO 15118.

[24] Alma L. Juarez-Dominguez (Ed.). 2008. *Feature Interaction Detection in the Automotive Domain: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering.*

[25] Alma L. Juarez-Dominguez, Nancy A. Day, and Jeffrey J. Joyce. 2008. Modelling feature interactions in the automotive domain. In *Proceedings of the 2008 international workshop on Models in software engineering*, Joanne Atlee (Ed.). ACM, New York, NY, 45.

[26] Dirk O. Keck and Paul J. Kuehn. 1998. The feature and service interaction problem in telecommunications systems: a survey. *IEEE Transactions on Software Engineering* 24, 10 (1998), 779–796.

[27] Willett Kempton and Jasna Tomić. 2005. Vehicle-to-grid power implementation: From stabilizing the grid to supporting large-scale renewable energy. *Journal of power sources* 144, 1 (2005), 280–294.

[28] Jihyun Lee, Sungwon Kang, and Danhyung Lee. 2012. A Survey on Software Product Line Testing. In *Proceedings of the 16th International Software Product Line Conference - Volume 1 (SPLC '12)*. ACM, New York, NY, USA, 31–40.

[29] Ivan do Carmo Machado, John D. McGregor, Yguaratã Cerqueira Cavalcanti, and Eduardo Santana de Almeida. 2014. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology* 56, 10 (2014), 1183–1199.

[30] Urtzi Markiegi. 2017. Test Optimisation for Highly-Configurable Cyber-Physical Systems. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*. ACM, 139–144.

[31] Bryan J. Muscedere, Robert Hackman, Davood Anbarnam, Joanne M. Atlee, Ian J. Davis, and Michael W. Godfrey. 2019. Detecting Feature-Interaction Symptoms in Automotive Software using Lightweight Analysis. In *SANER '19*, Xingyu Wang, David Lo, and Emad Shihab (Eds.). IEEE, Piscataway, NJ, 175–185.

[32] Sebastian Oster. 2005. Feature Model-based Software Product Line Testing. *International Workshop on Software Product Line Testing* 49, 12, 78–81.

[33] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software product line engineering: foundations, principles and techniques.* Springer Science & Business Media.

[34] Brian Prasad. 1997. Analysis of pricing strategies for new product introduction. *Pricing Strategy and Practice* 5 (1997), 132–141.

[35] pure systems. 2019. pure::variants: Variant Management with pure::variants. www.pure-systems.com/products/pure-variants-9.html

[36] Jens Schmutzler, Christian Wietfeld, and Claus Amtrup Andersen. 2012. Distributed energy resource management for electric vehicles using IEC 61850 and ISO/IEC 15118. In *Vehicle Power and Propulsion Conference (VPPC)*. IEEE, 1457–1462.

[37] Ken Schwaber and Mike Beedle. 2002. *Agile software development with Scrum.* Prentice Hall, Upper Saddle River, NJ.

[38] Vanessa Stricker, Andreas Metzger, and Klaus Pohl. 2010. Avoiding Redundant Testing in Application Engineering. In *Software Product Lines: Going Beyond*, Jan Bosch and Jaejoon Lee (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 226–240.

[39] T. F. Bowen, F. S. Dworack, C. H. Chow, N. Griffeth, G. E. Herman, and Y. -. Lin (Eds.). 1989. *The feature interaction problem in telecommunications systems: Seventh International Conference on Software Engineering for Telecommunication Switching Systems, 1989. SETSS 89.*

[40] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *Comput. Surveys* 47, 1 (2014), 1–45.

[41] Len Wozniak and Paul Clements. 2015. How automotive engineering is taking product line engineering to the extreme. In *Proceedings of the 19th International Conference on Software Product Line*. ACM, 327–336.