Model-based generation of test scripts across product variants: An experience report from the railway industry

Alessio Bucaioni1 | Fabio Di Silvestro2 | Inderjeet Singh3 |
Mehrdad Saadatmand4 | Henry Muccini5

Abstract
Software product line engineering emerged as an effective approach for the development of families of software-intensive systems in several industries. Although its use has been widely discussed and researched, there are still several open challenges for its industrial adoption and application. One of these is how to efficiently develop and reuse shared software artifacts, which have dependencies on the underlying electrical and hardware systems of products in a family. In this work, we report on our experience in tackling such a challenge in the railway industry and present a model-based approach for the automatic generation of test scripts for product variants in software product lines. The proposed approach is the result of an effort leveraging the experiences and results from the technology transfer activities with our industrial partner Alstom SA in Sweden. We applied and evaluated the proposed approach on the Aventra software product line from Alstom SA. The evaluation showed that the proposed approach mitigates the development effort, development time, and consistency drawbacks associated with the traditional, manual creation of test scripts. We performed an online survey involving 37 engineers from Alstom SA for collecting feedback on the approach. The result of the survey further confirms the aforementioned benefits.

Keywords
automation, model-based software engineering, product line engineering, testing

1 | INTRODUCTION

Nowadays, many industrial domains are increasing their demand for customized software-intensive systems as a way of addressing market needs, regional standards, certifications, software, and hardware requirements. To meet this demand, both researchers and practitioners in software engineering have proposed the use of software product line (SPL) engineering (SPLE).1 SPLE focuses on developing a set of software-intensive systems, sharing a common, managed set of features, from a common set of core artifacts2 (e.g., requirements and test cases). SPLE uses two development processes: the domain engineering and application engineering processes. The former focuses on the creation of the SPL platform through the identification of common and variable features and the development of domain artifacts realizing such features. The latter focuses on the derivation of individual systems based on the SPL platform identified in the domain engineering process.2 One of the main benefits introduced by SPLs is that all the development artifacts can be systematically organized and reused.3 This translates in a shorter time to market and increased quality hence in lower development cost.3

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.

© 2022 The Authors. Journal of Software: Evolution and Process published by John Wiley & Sons Ltd.

J Softw Evol Proc. 2022;34:e2498.
https://doi.org/10.1002/smr.2498
Although SPLs introduce several benefits in the software development process, it also brings in new challenges especially in those domains relying on cyber-physical systems such as railway, automotive, and so forth. One of these challenges relates to the dependencies of the software artifacts implementing the common and variable features and the underlying electrical and hardware systems. Some of the software artifacts realizing the SPL platform identified in the domain engineering phase may not be directly reused as the underlying electrical and hardware systems may differ. To clarify this, let us consider a real-world scenario from the railway domain. Alstom SA (Alstom) is a French multinational rolling stock manufacturer operating worldwide in the railway domain. Currently, Alstom in Sweden is producing a family of five electric unit trains for passengers transportation for the British Market: the Aventra family.

The Aventra family is developed using SPLE, and its SPL platform consists of a large number of basic (e.g., driver cabin activation, doors activation) and advanced functionalities (e.g., safety-related and train re-configurations). The Traction/Brake Control function (TBC) is one such functionality and is responsible for transmitting the driver inputs to the brake and propulsion systems by operating on the train communication channels, which are called signals. TBC alone is realized using a set of software artifacts spanning the whole life cycle (e.g., requirements and testing) and having different levels of abstractions (e.g., functional and technical). The software functional artifacts implementing the TBC share the same logic and they can be reused across the whole Aventra family since they operate on functional and logical variables or signals. The software technical artifacts, such as the test scripts, share the same logic, but cannot be reused for all the product variants in the Aventra family as they operate on concrete signals, which may differ due to different train architectures. Hence, different test scripts, accounting for different train signals, need to be created for each product variant in the family.

Traditionally, such a challenge was dealt with the so-called opportunistic reuse of test artifacts strategy. This strategy implies creating test scripts for one product variant of the family and replicating these scripts for the remaining product variants in the family. Then, each replica is manually modified so to account for possible product differences. Although widely adopted, the opportunistic reuse strategy carries several drawbacks, including:

- high development effort: manually modifying software artifacts is not only undesirable but often unfeasible for industrial SPLs;
- high error proneness: manual changes are more likely to introduce errors and replicating artifacts across the SPL may propagate them;
- consistency: manual changes make it difficult to keep artifacts consistent as changes to the original artifact need to be explicitly propagated to the replicas.

In this paper, we report on our experience in tackling the practical challenge of developing technical software artifacts stemming from functional software artifacts. Together with our industrial partner, Alstom, we built on our previous work and defined a model-based approach able to automatically generate test scripts starting from test case descriptions. The proposed approach uses a set of metamodels, a domain-specific language (DSL) and a model transformation. We applied the proposed approach on the Alstom Aventra SPL and used the results of this application for evaluating its industrial applicability and efficacy. Based on our evaluation results, we concluded that the proposed approach can mitigate the development effort, error proneness and consistency drawbacks typical of the opportunistic reuse strategy. We noticed that the proposed approach requires the manual development of a fewer number of software artifacts already for SPLs containing three product variants and two functionalities.

Eventually, we used an online survey involving 37 Alstom engineers for evaluating the relevance, acceptance, cost, benefits and return of investment (ROI) of the proposed approach.

The remainder of this paper is organized in two main parts. The first part discusses the technical solution to the automatic generation of test scripts in SPLs, and it consists of Sections 2–4. In particular, Section 2 describes the proposed approach and its main steps. Section 3 presents the application of the approach on the Aventra family, and Section 4 presents related work. The second part of this paper discusses the successful technology transfer to Alstom, and it consists of Sections 5 to 7. In particular, Section 5 describes the research methodology that we used for carrying out this research effort and the technology transfer. Section 6 describes potential threats to validity and related mitigation strategies. Section 7 discusses the relevance, acceptance, cost, benefits, and return of investment (ROI) of the proposed approach and lessons learnt. Section 8 concludes the paper with final remarks and possible future work.

## 2 | A MODEL-BASED APPROACH FOR THE AUTOMATIC GENERATION OF TEST SCRIPTS

In this section, we describe the proposed approach for the automatic generation of product variants test scripts stemming from common test cases. Figure 1 provides a graphical representation of the approach in terms of its main steps (marked with black-circled numbers in the figure), which are as follows: SPL description, product description, links specification, test cases description, and test scripts generation. These five steps can
be grouped into two conceptual phases: Steps 1 to 3 belong to the set-up phase (red box in the left half of Figure 1), while Steps 4 and 5 belong to the execution phase (gray box in the right half of Figure 1). The steps in the set-up phase aim at preparing the artifacts for the automatic generation, while the steps in the execution phase carry out the generation itself. In the remainder of this section, we describe each step of the proposed approach. For each of these steps, we present and describe the enabling artifacts. All the artifacts can be accessed at https://github.com/fabiodisilv/Model-Based_Test_Generator_SPL.

2.1 | SPL description

SPL description is the first step of the proposed approach and should be performed during the set-up phase. The goal of this step is to provide for a functional representation of the SPL functionalities, for example, TBC, driver cabin activation, and doors actuation. To support the creation of these representations, we defined the SPL metamodel (SPLmm). SPLmm is a metamodel enabling the representation of an SPL, its common functionalities, and their signals. Figure 2A provides a class diagram representation of SPLmm. The root metaclass is Family, which acts as a container and has two attributes: name and description. Family contains one or more GenericFunction metaclasses, which specify common functionalities of the SPL. A GenericFunction class has two attributes namely name and description and contains one or more GenericStep metaclasses. GenericStep abstract metaclasses represent the functional inputs and outputs of a functionality and have one attribute, name. A GenericStep metaclass can be specialized into a GenericInput or a GenericOutput; these represent the functional input or output of a functionality, respectively. Figure 2B depicts a model created using the SPLmm. The model depicts a family of products called SPL, which in turn contains a functionality named Function1. This function operates on one input, Input1, and one output, Output1.

SPLmm offers a lightweight instrument for capturing functional information about a SPL. It can be easily integrated with any development process and requires limited to no prior training. It should be noted that we extend our approach so other formalisms can be used in this stage. For instance, engineers can use feature modeling, which is an established formalism used for capturing functionalities (features) and their dependencies. Feature modeling has a well defined syntax and semantics and is supported by both open-source and commercial tools such as FeatureIDE. However, the use of other formalisms may affect steps number three and five of the proposed approach, as we discuss in the remainder of this section.

We developed SPLmm, and all the other metamodels used by the approach, as Ecore models within the Eclipse Modeling Framework (EMF).
2.2 Products description

Products description is the second step of the proposed approach and should be performed during the set-up phase. The goal of this step is to capture product variants belonging to the SPL together with their signals involved in the functionalities modeled in the first step. To support such a task, we defined the Products metamodel (Pmm), which is a language for designing product variants in terms of their functionalities and signals. Figure 3A depicts the class diagram for the Pmm. The root metaclass is Family. Family has two attributes, name and description, and contains one or more Product metaclasses. A Product metaclass specifies one product variant belonging to the SPL and has one attribute, name. A Product metaclass refers to one or more ProductSpecificFunction metaclasses, which are used to represent the product variants versions of common, functional SPL features. A ProductSpecificFunction metaclass has one attribute, name, and refers to one or more ProductSpecificStep and one or more Product metaclasses. A ProductSpecificStep abstract metaclass represents a product variant step composing a functionality, and it can be specialized into ProductSpecificInput and ProductSpecificOutput. A ProductSpecificStep metaclass refers to a Signal, where the Signal metaclass represents the product variant signal implementing a GenericInput or GenericOutput. Figure 3B shows an example of a model built using Pmm. The model depicts a family called SPL containing one product named Product1. This product features a functionality called SpecificFunction1, which uses one input, System1.Signal1, and one output, System1.Signal2. In turn, these refer to the concrete product variant signals Signal1 and Signal2, respectively.

The Pmm metamodel offers a structured way of capturing information on the specific architectures of product variants in an SPL. This is a crucial step for enabling the automatic generation of test scripts. We extended the proposed approach such that other notations can be used in this step. For instance, engineers might use configuration files containing the list of signals of each product variant. Similar to the previous step, the use of other formalisms may affect steps number 3 and 5 of the proposed approach. We discuss this in the remainder of this section.

2.3 Links specification

Link specification is the third step of the proposed approach and should be performed during the set-up phase. The goal of this step is to relate functionalities and signals of product variants of the SPL (developed in the second step) to their functional representations (developed in the first step). To support this task, we created the Weaving metamodel (Wmm), which allows engineers to specify models capturing links between elements of models built using SPLmm and Pmm. The information captured using Wmm is used for the automatic generation of executable test scripts. Figure 4A depicts the class diagram of the Wmm. The root metaclass is Weaving, which has two attributes, name and description, and contains one or more FunctionLink metaclasses, which are used for modeling the links. FunctionLink metaclasses type one ProductSpecificFunction to one GenericFunction and contain a list of InputLink and OutputLink metaclasses. The former is used to type ProductSpecificInput to GenericInput elements while the latter for typing ProductSpecificOutput to GenericOutput elements. Figure 4B shows an example of a model built using Wmm. The model describes links between the models depicted in Figure 3B and Figure 2B. In particular, the function link element links the product specific function SpecificFunction1 to the generic function Function1. Similarly, the input and output link elements link the specific input and output System1.Signal1 and System1.Signal2 to the generic steps Input1 and Output1, respectively.

Because Wmm links elements of SPLmm and Pmm, it can only be used for specifying relations between models built using these metamodels. If other notations are used for performing steps 1 and 2 of the proposed approach, Wmm cannot be used. To support the specification of links when these SPLmm and Pmm are replaced by other notations, such as feature modeling and configuration files, we provide an alternative metamodel named Lmm, which allows for the specification of simple links. Figure 5 shows its class diagram representation. The root metaclass is...
Weaving, which has two attributes, name and description, and contains one or more Link metaclasses. These are used for modeling the links. Link metaclasses have one attribute, GenericSignal, and contain one or more ProductSignal metaclasses. These contain two attributes being Product and Signal. The main difference between Wmm and Lmm is that the elements of Lmm can not be typed using elements of SPLmm and Pmm. Rather they are typed to strings, where the strings match the name of, for example, generic signals, products, and so forth. One main consequence of this is that the steps one and two of the approach may be skipped if different notations (than SPLmm and Pmm) are used. When creating a model using Lmm the engineers will not be able to use the information captured in the first two steps. This does not affect the automatic generation phase as we see in the next section. However, it has a great impact on the understandability, traceability and completeness of the approach itself. We discuss this in Section 7.

2.4 | Test case description

Test case description is the fourth step and needs to be performed during the execution phase. The goal of this task is to create test cases describing the checks to be performed on the common SPL functionalities as described in the first step. To support this task, we created the test case DSL (TcDSL). TcDSL enables the definition of test cases on the shared functionalities captured during the first step of the approach. Currently, TcDSL allows performing positive, unit testing. Listing 1 shows an excerpt of the TcDSL definition describing its main concepts. TcDSL

---

We have developed TcDSL using the Xtext programming languages development framework (https://www.eclipse.org/Xtext/).
allows for the definition of test cases by specifying the test case name and the functionality to test (line 2 of Listing 1) as captured by the artifacts in the first step.

The body of a test case is composed of a list of operations interacting with the inputs and outputs of the functionality. Currently, the supported operations are set, force, unforce, and check (line 4 of Listing 1). Lines 5 to 6 of Listing 1 show the behavior of the set and check operations, respectively. The set operation is responsible to specify the value of an input or output, while the check operation is responsible for controlling their value within a given timeout. It might happen that a test case or some of its operations do not apply to a given variant of an SPL. In this case, TcDSL allows for adding exceptions as shown by lines 5 and 6 of Listing 1. Listing 2 shows an example of test cases defined using TcDSL namely Check_OpenDoor and Check_OpenDoor_Exception.

Both test cases refer to the common feature OpenDoor responsible for operating train doors. Check_OpenDoor performs two operations being a set and a check. The set operation specifies the values of DoorLocked to False, while the check operation tests that DoorState is set to OPEN within a timeout of 5000 ms. Check_OpenDoor_Exception defines an exception on the Check_OpenDoor test case for product_A (line 5 of Listing 2).

### 2.5 Test script generation

Test script generation is the last step of the proposed approach and it should be performed during the execution phase. This step is responsible for the automatic generation of executable test scripts starting from test cases. This step is entrusted to a model to text transformation named Test Script generation Transformation (TST). A model transformation can be seen as a software program that manipulates an input or source model into an output or target model. Formally, TST can be described with the following function:

\[ TST: P, W, Tc \rightarrow n \times Ts \]

TST takes as inputs the model of the SPL SPL, the model P representing a number n of individual products of the SPL, the weaving model W and the test case Tc specified using TcDSL. Starting from these inputs, TST produces n test scripts Ts, one for each of the n products modeled using P. To this end, TST uses the following mapping rules:

- **P2Ts** creates a test script for each product in the SPL.
- **Tc2Method** creates a method in the test script for each specified test case.
- **Operation2Statement** creates a statement in the test script method for each operation in the test case.
• Signal2Parameter: translates each input/output of a test case operation into a signal. Besides, it marks the signal as the parameter of the test script statement.

TsT accounts for the exceptions defined using the TcDSL by skipping the products specified for the exceptions. We implemented TsT using a template-based technology called Acceleo. Acceleo allows the definition of a model transformation as a mix of static and dynamic elements. Static elements are expressed in the syntax of the target model and will not be changed at transformation time. Dynamic elements represent placeholders, which will be replaced with elements from the source/target models at transformation time. As test scripts in Alstom are written in C#, TsT uses static elements from the C# programming language. Listing 3 shows an excerpt of TsT implementing part of the P2Ts (line 1) and Tc2Method (line 4) rules. Besides it shows an example of static elements from C# in line 4.

In particular, Line 1 creates the C# file for the product and assigns it a name, which is the concatenation of TestSuite and the specific product name. Line 2 iterates on all the test cases defined in the test case descriptions (described in the previous step using the TcDSL). For each of these, it creates a C# function and assigns it the name of the test case (Line 4). Line 5 iterates on all the steps defined in the test case descriptions with the aim of translating them in C# statements. It is important to remark that, in the case Lmm is used in the third step of the approach, the transformation TsT can not be described by the above function. In this case, TsT is described by the following one:

\[ TsT < W, Tc > \rightarrow n \times Ts \]

As explained earlier, this is a consequence of Lmm not being able to use the structured information captured by SPLmm and Pmm.

3 | THE AVENTRA FAMILY: A USE CASE FROM THE RAILWAY DOMAIN

As described in Section 1, the Aventra SPL is a family of five electric unit trains for passengers transportation specifically designed for the British market. The trains are: LOT, EAA, SWR, WML and C2C. All the trains within the Aventra SPL share a considerable number of functionalities. When we applied the proposed approach on the Aventra SPL, the smoke tests for this family identified more than 18 functionalities shared among all the trains. In Section 1, we introduced one shared functionality called TBC, which is responsible for transmitting the driver inputs to the brake and propulsion system. In the remainder of this section, we demonstrate the application of the proposed approach on the Alstom Aventra SPL.

For the sake of brevity, we focus on two shared functionalities, TBC and Driver Cabin Activation, and three trains, LOT, EAA and SWR. The interested reader can access the full implementation of the Aventra use case at https://github.com/fabiodisilv/Model-Based_Test_Generator_SPL.

The first step of the proposed approach is to capture information about the Aventra family and its features. We did that, using SPLmm and Figure 6A shows a tree-based representation of an excerpt of the corresponding model. The model describes the Driver Cabin Activation and TBC functionalities (named as Function ActivateCab and Function TBC_Response in the figure) along with their steps (named as, e.g., Input MASTER_HW11_INPUT1, Input TBC_Demand_Level_Validity_1, in the figure).

The second step of the proposed approach is to capture information about product variants and their specific signals. For this task, we used Pmm and Figure 6B shows a tree-based representation of an excerpt of the corresponding model. The model describes the EAA, LOT and SWR trains (named as Product EAA, Product LOT and Product SWR in the figure) along with their functionalities (named as, e.g., Product Specific Function ActivateCab_EAA, Product Specific Function TBC_Response_EAA, etc., in the figure). In addition, Figure 6B shows some of the train specific steps and signals involved in these functionalities (e.g., Product Specific Input SYS1.EAA-SWR-DEM_LVL_VALID_1-EAA-SWR, Product Specific Input SYS1.EAA-SWR-DEM_LVL_1-EAA-SWR, Signal EAA-SWR-DEM_LVL_VALID_1-EAA-SWR, Signal EAA-SWR-DEM_LVL_1-EAA-SWR). Here, we can see how Pmm allows capturing train differences. For instance, Figure 7 shows that the EAA and LOT trains use four inputs and four outputs for implementing the Activate Cab feature (ActivateCab_EAA and ActivateCab_LOT), while the SWR train uses two inputs and four outputs (ActivateCab_SWR). Another difference is that EAA and LOT use different inputs and outputs.

```
[ file ('TestSuite'.concat(aProduct.name.concat(' .cs')),false , 'UTF-8')] 
[ for (aTestCase : TestCase | aTestSuite.testCases /] 
  ...
  public void [aTestCase.name /]){ 
    [for (aStep : Step | aTestCase.steps )]
      ...
    [/for]}
```

LISTING 3  Excerpt of TsT transformation implementing P2Ts and Tc2Method rules
The next step of the proposed approach is about capturing the links between the product variants functionalities and signals and their generic representation. As we used SPLmm and Pmm in the first two steps, we performed this task using the Wmm. Figure 8 shows some excerpts of the corresponding model. In particular, Figure 8A shows two Function Link elements, which type the generic TBC and Activate Cab functionalities to their product variants version. For instance, Figure 8B shows that the generic function ActivateCab is linked to Product Specific ActivateCab_EAA, Product Specific ActivateCab_LOT and Product Specific ActivateCab_SWR. Figure 8C shows a further excerpt of the weaving model linking the generic input Input MASTER_HW11_INPUT1 of Function ActivateCab to the specific inputs Product Specific Input SYS.EAA-SWR-HW11_SIGNAL95_INPUT1-EAA-SWR and Product Specific Input SYS.LOT-HW11_SIGNAL92_INPUT1-LOT of ActivateCab_EAA, ActivateCab_LOT and ActivateCab_SWR, respectively (Figure 6B).

The fourth step is the specification of test cases for the SPL functionalities as captured in the first step. This step is performed using TcDSL. Listing 4 shows a test case for the Activate Cab functionality named Check_ActivateCab and two test cases for the TBC_Response functionality named Check_TBCResponse1 and Check_TBCResponse3. Check_ActivateCab consists of two force and four check operations. Check_TBCResponse1 and Check_TBCResponse3 consists of two force and two check operations each.

The semantics of the operations is defined in TcDSL. Note that the operations defined in the test cases operates on the generic signals as captured in the first step.

The last step of the proposed approach is the generation of executable test scripts from the test cases specified in the previous step. The generation is entrusted to the TsT transformation described in Section 2. As described in Section 2, the test scripts for the Aventra family are specified using C#. The execution of TsT produces one C# file for each product variant in the Aventra family. Each of these files contain the set of C# instructions implementing the logic of the test cases defined in the previous step. Listings 5 describes a portion of the generated C# file for the EAA train containing the Check_ActivateCab, Check_TBCResponse1 and Check_TBCResponse3 test cases described in Listing 4. It can be seen how the linking information captured by the model in Figure 8 is used by the TsT for the generation of the test scripts. For instance, the generic input MASTER_HW11_INPUT1, used in the definition of the Check_ActivateCab test case in Listing 4, is substituted with the EAA specific signal EAA-SWR-HW11_SIGNAL95_INPUT1-EAA-SWR in Listing 5 using the information captured by the model in Figure 8C.
Our approach lies at the intersection of model-based testing automation and software product line testing in the context of the railway domain. While much research has been performed on the use of model-based techniques for validation and testing, and product line applications testing alone, this section brings the reader attention to the approaches most directly related to our proposal. Scippacercola et al. present an approach for model-in-the-loop testing of a railway interlocking system. Exploiting model-driven architecture (MDA), the authors define a computation independent test model (CIM), to express interactions as properties and conditions related to the environment. A model of the expected behavior of the system components is specified via a black box platform independent test model (BB-PIT). Test cases are specified by sequence, activity, and state machines diagrams using the unified modeling language (UML) testing profile (UTP). The approach is tested on the Prolan Monitor component, which is part of a railway interlocking system. Similarly to us, Scippacercola et al. use different domain-specific languages, a model-driven engineering approach, and computation independent test models. However, while their focus is on a specific product, our focus is on a product line approach to automatically generate product-specific test scripts from a product line. Other approaches proposing the use of UTP for specifying test cases for both static (structural) and dynamic (behavioral) aspects of software systems are those in previous studies.

Another approach using UTP while not applying it on the railway domain is the one by Bagnato et al. that describes an industrial application of UTP within the field of future internet application. Haxthausen and Peleska propose a bounded model checking approach, combined with inductive reasoning, and model-based testing in the railway domain. For a given product line, the formal (generic) model is initially created and checked for well-formedness. Then, test strategies are extracted from the generic models. The configuration data specific to concrete interlocking system components is successively used to generate concrete model-based integration tests. Similarly to our approach, Haxthausen and Peleska take a product line-based approach to test a railway system. Differently, while their main focus is on integrating formal verification with model-based testing, our scope is specific to model-based testing with a fully automated generation of test scripts from the product line model.

Łukasik and Nowakowski propose a TTCN-3 based approach for dynamic testing of railway interlocking systems. TTCN-3, with Abstract Syntax Notation One (ASN.1), is used to model and test a railway interlocking communication interface. Railway safety requirements are mapped
to TTCN-3 test cases successively implemented through a TTCN-3 test system. The work by Łukasik and Nowakowski is profoundly different from ours: while we both use a model-driven approach, our main goal is on the automatic generation of test scripts from shared SPL, in contrast with their approach envisioning the use of TTCN-3.

In line with the model-driven engineering approach to test script generation presented in our paper, we may also cite the ETSI Test Description Language (TDL). TDL is a language for the specification of test descriptions and the presentation of test execution results, therefore related to our TcDSL.

Other approaches, while not explicitly focusing on software testing, provide model-driven engineering approaches for the verification and validation of safety-critical systems in the railway domain. Petry et al. performed a mapping study on model-based testing (MBT) of software product lines. The main findings related to our approach can be summarized as follows.

- **Application domains**: software and automotive are reported to be the most frequent domains for SPL testing. No specific reference is made to railways systems. This remarks the novelty of using MBT on railways systems product lines.
- **Automation**: in line with our paper, the vast majority of MBT approaches for SPL testing are fully- or semi-automated. This remarks the necessity of fully automated approaches for SPL testing.
Artifacts: state machines, transition diagrams, and functional diagrams are most frequently used to MBT SPLs. None of the primary studies seem to use domain specific languages for test case definition. This remarks another innovative aspect of our approach.

Variability: 68.1% of studies treat variability during any MBT of SPLs activities. This is in line with our approach.

Evaluation: case studies are frequently performed to evaluate existing approaches. This is the case for our research effort too.

Reuys et al. propose a model-based approach to test case derivation in the system test of software product lines.22 The approach is known as Scenario based Test case Derivation (ScenTED) and requires test artifacts for the whole family to be designed for extensions to cover the variability of each product. What is more, within ScenTED test cases are automatically generated from system models such as the UML activity diagrams or sequence diagrams. Similar to ScenTED, our approach generates test artifacts starting from a structured representation of information families and products within families. However, our approach does not rely on behavioral system models, but on lightweight models. This makes our approach more flexible and suited for those contexts where behavioral information is not available.

Lochau et al. illustrate an application of the so-called delta-oriented testing technique, which is an incremental testing technique relying on state machines describing the products behaviors.23 The approach first generates test artifacts based on the state machines representing a product. Later, it evolves the generated test artifacts based on modifications calculated on the state machines. Dukaczewski et al. present another delta-oriented testing technique, which replaces state machines with textual requirements.24 Similar to the above delta-oriented testing techniques, the goal of our approach is to enhance (automatic) test artifacts generation. However, our proposed approach differs from the above-mentioned ones as executable test scripts are generated for each product starting from a single test case for the whole family.

Asaithambi and Jarzabek propose an approach known as the generic adaptable test cases for SPLs.25 The approach aims at reducing the number of test cases needed for testing all the products within a given family. Test cases are generated after analyzing different assets of the SPL, including already existing test cases. The analysis aims at spotting and removing duplicated test cases. What is more, similar test cases are generalized and grouped. The approach by Asaithambi and Jarzabek differs from the proposed approach as the analysis and the test cases generation are done manually. Besides, the approach has not been applied to any real-world use case.

Iber et al. present an approach, which use the interplay of a DSL and UTP.14 Even though the enabling artifacts of our approach and the one by Iber et al. are similar: formal representation of the system, domain specific language for the test case definition, and a transformation to generate the real test scripts, the two works target different applications and use cases. In Iber et al.,14 the authors’ focus is on the definition of a textual DSL to overcome the limitations of the UML Testing Profile defined as “too broad in order to solve specific problems.” Our research aims at providing a lightweight model-based approach to increase the testing efficiency in software product lines. The different end goals bring the two works to have dissimilarities for example our approach does not require a UML artifact modeling the whole system, but the focus is only on the common/product specific features and how they are distributed among the SPL products. Another noteworthy difference is the transformation, on their side only to generate test scripts in the target programming language, on our side to generate test scripts tailored for the product.

5 | RESEARCH METHODOLOGY

There exists several research methodologies for researching in software engineering. We defined, developed, and validated the proposed approach following an adaptation of the research methodology introduced in Gorschek et al.26 This methodology focuses on preparing and maximizing the technology transfer between academia and industry. To this end, it uses an iterative process emphasizing the evaluation of the technology to be transferred using a multi-step validation process. Figure 9 shows a graphical representation of the adopted research methodology in terms of its main steps. The entire process lasted for 18 months and involved all the five researchers. The first step is the elicitation of the industrial needs. This step was initiated by the Alstom representatives and involved all the researchers. Note that in our case, the customer was the Aventra engineering team and that Alstom has a long-standing collaboration with all the researchers involved in this effort. The main need
described by the Alstom representatives was to improve the development time and effort associated with the creation of test scripts for the product variants of SPLs. A side need of Alstom was also to formally capture and represent variability within test artifacts. Starting from the identified need and using a review of the literature, we derived the research goal that was to automate the creation of test scripts for SPL product variants. Once we defined the goal, we started to survey the literature for possible solutions. Considering that Alstom had an interest in shifting into model-based solutions, we proposed the definition of a model-based approach. Other candidate solutions were the use of specific tools supporting feature modeling or the use of data sets. Once we identified, defined and developed our model-based process, we proceeded with its validation. The validation process leveraged by this methodology consists of three steps, two of which happen within industrial settings. Validation in academia is the first validation step and, as the name suggests, is performed in academic settings by researchers. This validation uses methods such as workshops and academic use-cases. In our case, we validated the proposed approach using academic use-cases from previous projects. The approach was applied on the selected use-cases by the academics involved in this research. Static validation is the second validation step and the first step to be conducted within industrial settings. This step revolves around dedicated meetings where the proposed solution is presented to selected representatives from the company. The aim is to collect early feedback on the industrial feasibility and applicability of the solution before it is tested on real, industrial use cases. In our case, we conducted dedicated online workshops with the engineers from the Aventra integration team at Alstom. In these workshops, we first described the proposed approach and its application on the academic use cases, and then collected practitioners feedback and comments. We used the elicited comments for improving the proposed approach before the last validation step that is dynamic validation. This step requires the use of real-world examples as a way for testing the proposed solution. To this end, we executed the proposed approach on the Alstom Aventra SPL. The execution of the approach on the Aventra SPL was entrusted to the Alstom engineers involved in this research and it was carried out at Alstom premises in Västerås, Sweden. More details on this step are provided in Section 3. We extended the research method from Gorschek et al, with an additional step, namely, an online survey. This step aimed at collecting practitioner opinions on specific aspects of the proposed approach being: relevance, acceptance, cost, benefits and ROI. To this end, we created an online survey, which we sent to the whole Aventra team. The survey contained 10 questions including closed- and open-ended ones. We discuss the results of the survey in Section 7. The interested reader can access the complete survey and results at https://github.com/fabiodisilv/Model-Based_Test_Generator_SPL.

6 | THREATS TO VALIDITY

In this section, we discuss and classify potential threats to validity as well as the adopted mitigation strategies according to the scheme proposed in Runeson and Höst.

6.1 | Internal

Threats to internal validity affect the ability to draw correct relationships between treatment and outcome. To mitigate possible threats to internal validity during the dynamic validation, we used a real-life use case coming from our industrial partner without modifying it.

When developing the proposed framework, we have made an effort to not introduce new tools or technologies and for limiting the impact of tool performance. We have built the proposed framework incrementally, carefully testing each increment. To mitigate possible threats to internal validity during the static validation, we selected practitioners with proven and extensive experiences in testing, MDE and railway domain. We have made an effort to ask questions in a neutral way so as not to bias respondents with more positive answers in favor of the proposed approach.

6.2 | External

Threats to external validity affect the ability to generalize the results beyond the experiment settings. To mitigate such threats during the validation activities, we made an effort in selecting subjects with prior and proven experience in testing, MDE and railway domain. In addition, the involved practitioners had different nationalities and levels of experience. Hence, we believe that the results of the interviews are agnostic of the country of origin and the level of experience of the participants. We validated our approach using a real-life use case coming from our industrial partner, which we did not modify or alter. The validation on the use case has been carried out entirely at the Alstom premises in Västerås. Although the use case came from our business partner, the use of a single-use case may reduce the generalizability of the results. We are considering applying the proposed approach to further use cases as future work. The tools and technologies leveraged in this study have been discussed and used in several domains besides railway. Hence, the approach can be easily applied in other domains, although minor adjustments may be needed, for example, metamodels and model transformation.
6.3 | Construct

Threats to construct validity relate to the extent to which the setting of an experiment reflects the theory. The proposed framework has been validated on the Aventra SPL, which is a real system from Alstom. To overcome the possible mono-method bias, we are working on applying the proposed framework to additional use cases. We used dedicated workshops as the main means of (static) validation so to reduce evaluation apprehension and hypothesis guessing threats. The expert interviews have been opened by an informal discussion on the proposed approach and a questions and answers session for mitigating the risks of misunderstandings. Besides, it is worth mentioning that all the authors have a longstanding collaboration with the practitioners involved in the study and this has resulted in insightful feedback characterized by mutual trust.

6.4 | Conclusion

Threats to conclusion validity affect the ability to derive a correct conclusion from the relations between treatment and outcome. To minimize threats to conclusion validity, the proposed framework has been analyzed by the independent practitioners at our business partners. In addition, to show the correctness of the proposed approach, we compared the generated test scripts with the hand-crafted ones. The comparison showed that the proposed approach was able to generate test scripts equivalent to those created manually using the opportunistic reuse of test artifacts strategy.

7 | DISCUSSION AND LESSONS LEARNED

The automatic generation of test scripts across product variants is one of the challenges hampering the full-fledged adoption of SPLE within Alstom. To tackle this challenge, we proposed a model-based approach and described its application on an industrial use case: the Aventra SPL. In this section, we discuss important aspects related to the proposed approach, such as relevance, acceptance, cost, and benefits. Some of these aspects are discussed considering the results of the online experts survey, which involved 37 engineers from Alstom and composed of the 10 questions reported in Table 1. All the engineers belonged to the Alstom Aventra team: 4 of them as managers, 7 of them as testers, and 26 as developers. We did not notice any correlation between roles and differences in opinions.

7.1 | Model-based approach versus opportunistic reuse

Traditionally, test scripts were developed using the opportunistic reuse of test artifacts strategy. The main drawbacks associated with this strategy are: high development effort, error proneness and consistency issues.

| Question                                                                 | Type          |
|-------------------------------------------------------------------------|---------------|
| In the current manual approach, what are some of the issues you've faced in developing test scripts or (re)using them for each new product variant and version? | Open-ended    |
| How important is it (for you and Alstom in general) to be able to automatically generate test scripts for each new product variant and version? | Close-ended (scale) |
| How well do you think the approach can be accepted to be used at Alstom? | Close-ended (scale) |
| How much do you think the proposed automatic approach fits (technically) with the existing Alstom inhouse tooling? | Close-ended (scale) |
| Which skills (potentially not available in the company) are required to follow this model-based testing process? | Open-ended    |
| Does this approach require a change in your organizational processes and structures? | Open-ended    |
| How do you rate the cost associated to the modeling phases of the approach; i.e., the effort and initial investment to build the models? | Close-ended (scale) |
| How do you rate the cost associated to the maintenance of the approach? | Close-ended (scale) |
| What are the main benefits that you think the approach can bring? | Open-ended    |
| What do you think the overall expected Return-On-Investment (ROI) of the approach could be [e.g., the overall benefits and gains versus cost]? | Open-ended    |
Concerning development effort, the opportunistic reuse strategy requires the manual creation of \( N_{\text{opportunistic reuse}} = P \times F \) artifacts, where \( P \) is the number of products and \( F \) is the number of features in an SPL. In fact, the opportunistic reuse strategy requires the creation of an artifact for each feature of each product of an SPL. If \( F \) is the number of features and \( P \) the number of products in an SPL, then the number \( N \) of artifacts to be created with the proposed approach is \( N_{\text{proposed approach}} = 3 + F \), in the case SPLmm and Pmm are used. In fact, regardless of the size of the SPL, the proposed approach requires creating three models for the first three steps. Besides, it requires the creation of a test script for each of the \( F \) features contained in the SPL. The number \( N \) of artifacts to be created with the proposed approach can be reduced to \( N_{\text{proposed approach}} = 1 + F \), in the case other notations are already used within the development process. Figure 10 compares the graphs representing the number of required artifacts from the proposed approach and the opportunistic reuse strategy. Figure 10A shows that the proposed approach requires a higher number of artifacts when the size of the SPL is relatively small. However, the proposed approach requires a fewer number of artifacts already for SPLs containing three product variants and two features (Figure 10B) or five product variants and one feature (Figure 10C). Previous studies reported that the average size of industrial SPLs consists of tens of products and hundreds of features.\(^{28,29}\) Our experience with Alstom confirmed this evidence. One may argue that the mere comparison of the number of required artifacts might be inaccurate as it does not take into account the complexity and size of the artifacts hence their development time. While this may be a valid concern, we found that it is not straightforward to empirically measure their development time as this may be biased by several factors (such as proficiency of developers) with different technologies. We are planning to investigate this aspect as future work.

With the proposed approach, the error proneness and consistency are mitigated and ensured by construction as test scripts are generated automatically starting from test cases using a model transformation. Hence, as long as the model transformation is correct, the generated test scripts are free from errors and inconsistencies. To discuss these points, we compared the test scripts generated by the proposed approach to test scripts written manually by Alstom engineers. Note that we considered the test scripts written manually by the Alstom engineers as the gold standard. We carried out the comparison using the `difflib` Python module.\(^{30}\) The comparison showed that the automatically generated test scripts for the EAA, LOT and SWR product variants were equivalent to the handcrafted ones. No functional differences were found. In fact, the only differences were the comments that we injected into the generated ones (e.g., lines 1 and 2 in Listings 5) as a way for improving the understandability and maintainability of the generated artifacts.\(^{31}\) The interested reader can access the comparison results, handcrafted and generated test scripts at https://github.com/fabiodisilv/Model-Based_Test_Generator_SPL, which can be used for running the similarity check.

### 7.2 Relevance and acceptance of the proposed approach

According to the survey results, 91.9% of the interviewed Alstom engineers answered that the possibility to automatically generate test scripts for each new product variant is either very important or important for them (Figure 11A). In particular, 75.7% of the respondents consider this possibility very important while 16.2% consider it important. Only 8.1% of the respondents found the automatic generation of test scripts to be fairly important. Most of the respondents agreed that “developing test scripts is easy, but it still takes a lot of time” and “human effort.” When asked about issues or limitations of previous approaches for test script generation, respondents confirmed that development effort, error proneness, and consistency are the most prominent ones. “With the current approach is difficult to keep track of different product variants due to a lack of formalism to represent them. This makes hard to retrieve project specific details needed for test scripts. Sometimes, test scripts fail due to
incorrect project specific signals specification leading to software deliverable delays.” “Test specifications need to be changed manually every time and it becomes tricky” as “small changes in signal naming conventions may cause large and complex changes in documentation, test scripts and reports.”

Most of the interviewed Alstom engineers (86.5%) found that the approach can be either very well accepted or well accepted within the company (Figure 11B). In particular, 35.1% of the respondents found that the approach can be very well accepted while 51.4% of the respondents found that it can be accepted. 13.5% of the respondents were unsure upon the acceptance of the proposed approach within Alstom. The main factors that may affect the acceptance of the proposed approach are lack of skills and possible changes in the organization processes and structures. Skills and knowledge on model-based development are by far the most cited skills that may be potentially lacking within Alstom. Other mentioned skills were automated software testing and programming skills. In this respect, the extension to the proposed approach presented in this work may help in mitigating the impact of such a lack as it replaces models and metamodels with other notations. One respondent highlighted that “The proposed approach is compatible with current processes although some (not meaningful) adaptations may be needed.” In particular, “the organisation needs to build upon the modeling and automated test scripts generation [...] for SPLs, it needs to be decided which percentage of the organisational resources need be allocated to domain engineering or application engineering deliverables.” In this respect, the proposed approach may help in saving organizational resources and improve their allocation.

7.3 Perceived cost of the proposed approach

Introducing a new approach possibly entailing a new process may lead to higher costs in the short term. Eventually, if the cost is perceived to be too high and exceeds the benefits brought by the approach, this can be rejected by a given organization. To understand the potential cost of the proposed approach, we asked Alstom engineers to rate the cost of its modeling phases and maintenance. In fact, based on our experience, these are the activities that may be more expensive in terms of skills and resources as they require some knowledge in model-based development. The majority of the respondents (56.8%) did not find the modeling phases of the approach to be particularly demanding (Figure 12A). However, 40.5% of the respondents thought the initial investment and effort of building the models is either high (32.4%) or very high (8.1%) (Figure 12A). Only 2.7% of the respondents found the cost associated with these activities low (Figure 12A). These results are not surprising and seem to be in line with those on the acceptance of the approach. In fact, Alstom engineers perceived the modeling activities as more demanding due to the potential lack of knowledge and skills in model-based development at the company. In this respect, the extension to the proposed approach presented in this work may help in lowering such a cost as it supports replacing some modeling activities.
While Alstom engineers seemed to agree on the cost of the modeling phases, they had a more divided opinion on the cost associated with the maintenance of the approach. More than one third of the respondents (35.1%) found this cost to be either low (21.6%) or very low (13.5%) (Figure 12B). However, 35.1% of the respondents also found this cost to be high (32.4%) or very high (2.7%) (Figure 12B). Almost a third of the respondents thought that this cost is affordable for the company (29.7%) (Figure 12B). It is true that the proposed approach does not require maintenance in absence of changes. The metamodels and the model transformation represent a one time effort and they can be potentially reused as long as the test scripts use the same information and are developed using the same programming language. On the other hand, in the case of changes to, e.g., the programming language used for the test scripts, the model transformation should be revised for being able to generate test scripts in the new target language. Note that the maintenance cost might be higher in the case the proposed approach is run with other notations rather than SPLmm and Pmm. In that case, the transformation needs to parse all the artifacts and extract the information that, in the case of SPLmm and Pmm, are stored in the models.

The above results are also in line with the engineer opinions on interoperability between the current Alstom tooling and the proposed approach (Figure 13). The majority of respondents found the proposed approach to be fitting the current technologies (67.5%). In particular, 24.3% of the responses found the approach to be very well fitting the current technology stack. Almost one third of the respondents were neutral (27%) while only 5.4% of the experts found the approach to be difficult to integrate.

While the data reported in this section comes from the 37 engineers from Alstom involved in the experts survey, future work is planned to collect objective data on costs and cost efficiency.

7.4 | Benefits and return of investment of the proposed approach

The main goal of this joint research effort was the definition of an automatic approach for the generation of test scripts across product variants that could lower the drawbacks of traditional reuse approaches. Earlier, we discussed how the proposed approach mitigates the development effort, error-proneness and consistency typical of the opportunistic reuse strategy. Besides these benefits, Alstom engineers identified additional improvements introduced by the proposed approach. We grouped the benefits identified by the respondents in categories, which are: cost-efficiency, quality, maintainability and reusability of the test artifacts. Concerning the cost-efficiency of the proposed approach, most of the engineers acknowledged that the automation introduced by the proposed approach could have a positive impact on the delivery pace and productivity of the projects: “the automation of test scripts generation would streamline the process and help save time (and hence money).” In addition, the proposed approach could facilitate the use of company resources more efficiently: “it will change the way test scripts are developed from a coding perspective to a configuration management perspective.” Engineers at Alstom are confident that, in the medium and long run, the proposed approach could reduce the development costs. “Even though the initial modeling phase is moderately heavy for large SPLs, on the middle long term, the approach is going to drastically decrease the costs [...] over time, the ROI shall be positive, improving over time and eventually reaching a point where the resources used to develop/maintain test cases tend to 0.”

Our findings and the aforementioned benefits of our proposed approach are also in line with a recent study, which reports on the benefits and challenges of adopting software product line engineering in industry. In particular, the study in previous study mentions reduced lead time for product development, testing, and increased confidence in the product as some of the benefits of adopting a product line engineering approach in industry. It also identifies, among other things, reuse recommendation, test reuse, variability-aware testing, and verification as some of the open challenges in this regard. The solution we have proposed in this paper for automated test script generation across product variants, in particular, addresses the test reuse and variability-aware testing and verification challenges, and therefore, can help industries in their transition towards adopting product line engineering.

At the moment of writing, Alstom Aventra SPL is near to project closure. We used the Aventra SPL as a proof of concept for the proposed approach. Based on the results and expert interviews, Alstom is planning to use the proposed approach in upcoming projects.

![Figure 13](https://example.com/figure13.png)

**Figure 13** Practitioner opinions on the fit of the proposed approach with existing in-house tooling
CONCLUSION AND FUTURE WORK

In this work, we tackled the challenge of automatically generating test scripts across product variants by introducing a model-based approach. After describing and exemplifying our approach, we showed its application on the Aventra train family from Alstom SA. The application showed that the approach is applicable in industrial settings and mitigates the development effort, error proneness and consistency drawbacks of the opportunistic reuse of test artifacts strategy for SPLs containing three products and two features. Using an online survey, we collected the opinions of 37 engineers from Alstom SA on the acceptance, relevance, cost, benefits and return of investment of the proposed approach.

Future work may encompass several directions. One possible direction is to evaluate the usability of the approach and its composing artifacts on different domains than railway. The results of this validation may be used for the refinement of the artifacts and for improving the generalisability of the approach itself. Another possible extension to the approach may encompass the refinement of its enabling artifacts to support the automatic generation of different development artifacts than test scripts. A further line of future work involves extending the validation of the approach. In particular, the approach might be applied and evaluated on other real-world use cases of different size and complexity.

ACKNOWLEDGMENTS

The work in this paper is supported by the Swedish Knowledge Foundation (KKS), through the project A-CPS, and by the Swedish Governmental Agency for Innovation Systems (VINNOVA), through the projects PANORAMA and XIVT. The authors would like to thank the Aventra integration team for the availability and the insightful discussions.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in https://github.com/fabiodisilv/Model-Based_Test_Generator_SPL.

ORCID

Alessio Bucaioni https://orcid.org/0000-0002-8027-0611

REFERENCES

1. Pohl K, Böckle G, Van Der Linden FJ. Software Product Line Engineering: Foundations, Principles and Techniques: Springer; 2005.
2. Metzger A, Pohl K. Software product line engineering and variability management: achievements and challenges. Future Softw Eng Proc. 2014;10: 70-84.
3. Frakes W, Kang K. Software reuse research: status and future. IEEE Trans Softw Eng. 2005;31(7):529-536.
4. Broy M, Gleirscher M, Merenda S, Wild D, Kluge P, Krenzer W. Toward a holistic and standardized automotive architecture description. IEEE Comput. 2009;42(12):98-101.
5. Da Mota Silveira Neto PA, De Carmo Machado I, Mc Gregor JD, De Almeida ES, De Lemos Meira SR. A systematic mapping study of software product lines testing. Inform Softw Technol. 2011;53(5):407-423.
6. Bucaioni A, Di Silvestro F, Singh I, Saadatmand M, Muccini H, Jochumsson T. Model-based automation of test script generation across product variants: a railway perspective. In: 2021 IEEE/ACM International Conference on Automation of Software Test (AST); 2021:20-29.
7. Beuche D, Dalgarno M. Software product line engineering with feature models. Softw develop Mag - Project Manag Programm Softw Test. 2006:78: 5-8.
8. Meinecke J, Thüm T, Schirer R, Benduhn F, Leich T, Saake G. Mastering Software Variability With Featureide: Springer; 2017.
9. Sendall S, Kozaczynski W. Model transformation: the heart and soul of model-driven software development. IEEE Softw. 2003;20(5):42-45.
10. Eclipse Foundation. Acceleo. Accessed: February 23, 2022. https://www.eclipse.org/acceleo/
11. Musset J, Julié E, Lacrampe S, et al. Acceleo user guide. Accessed: February 23, 2022. http://acceleo.org/doc/obeo/en/acceleo-2.6-user-guide.pdf
12. Bucaioni A, Mubeen S, Ciccozzi F, Cicchetti A, Sjödin M. Modelling multi-criticality vehicular software systems: evolution of an industrial component model. Softw Syst Model. 2020;19(5):1283-1302.
13. Scippacercola F, Pietrantuono R, Russo S, Zentai A. Model-in-the-loop testing of a railway interlocking system. In: International Conference on Model-Driven Engineering and Software Development; 2015:375-389.
14. Iber J, Kajtazovic N, Macher G, Höller A, Rauter T, Kreiner C. A textual domain-specific language based on the UML testing profile. In: International Conference on Model-Driven Engineering and Software Development; 2015:155-171.
15. Schieferdecker I, Dai ZR, Grabowski J, Rennoc A. The UML 2.0 testing profile and its relation to TTCN-3. In: IFIP International Conference on Testing of Software and Communicating Systems; 2003:79-94.
16. Baghato A, Sadovych A, Brosse E, Vos T. The OMG UML testing profile in use—an industrial case study for the future internet testing. In: 17th European Conference on Software Maintenance and Reengineering; 2013:457-460.
17. Haxthausen AE, Peleska J. Model Checking and Model-Based Testing in the Railway Domain: Springer; 2015.
18. Łukasik Z, Nowakowski W. Application of TTCN-3 for testing of railway interlocking systems. In: International Conference on Transport Systems Telematics; 2010:447-454.
19. Scippacercola F, Pietrantuono R, Russo S, Zentai A. Model-driven engineering of a railway interlocking system. In: 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD); 2015:509-519.

20. Amendola A, Becchi A, Cavada R, et al. A Model-Based Approach to the Design, Verification and Deployment of Railway Interlocking System: Springer; 2020.

21. Petry K, Oliveira Jr E, Zarzo A. Model-based testing of software product lines: mapping study and research roadmap. J Syst Softw. 2020;167:110608.

22. Reuys A, Kamsties E, Pohl K, Reis S. Model-based system testing of software product families. In: International Conference on Advanced Information Systems Engineering; 2005:519-534.

23. Lochau M, Schaefer I, Kamischke J, Lity S. Incremental model-based testing of delta-oriented software product lines. In: International Conference on Tests and Proofs; 2012:67-82.

24. Dukaczewski M, Schaefer I, Lachmann R, Lochau M. Requirements-based delta-oriented SPL testing. In: 4th International Workshop on Product Line Approaches in Software Engineering (PLEASE); 2013:49-52.

25. Asaithambi SPR, Jarzabek S. Generic adaptable test cases for software product line testing: software product line. In: Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity; 2012:33-36.

26. Gorshe T, Garre P, Larsson S, Wohlin C. A model for technology transfer in practice. IEEE Softw. 2006;23(6):88-95.

27. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng. 2009;14(2):131-164.

28. Bosch J. Product-line architectures in industry: a case study. In: Proceedings of the 21st International Conference on Software Engineering; 1999:544-554.

29. Nestor D, O’Malley L, Quigley A, Sikora E, Thiel S. Visualisation of variability in software product line engineering. In: First International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2007; 2007.

30. Python. Python difflib library. Accessed: February 24, 2022. https://docs.python.org/3/library/difflib.html#difflib.SequenceMatcher

31. Scalabrino S, Bavota G, Vendome C, Linares-Vásquez M, Poshyvanyk D, Oliveto R. Automatically assessing code understandability: how far are we? In: 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE); 2017:595-613.

32. Abbas M, Jongeling R, Lindskog C, Enou E, Saadatmand M, Sundmark D. Product line adoption in industry: an experience report from the railway domain. In: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A; 2020:1-11.

AUTHOR BIOGRAPHIES

Alessio Bucaioni is a software engineer currently working as assistant professor in computer science at Mälardalen University. He received his Ph.D degree from Mälardalen University in 2018. His research focuses on several aspects of the development of complex software-intensive systems from software architecture to model-driven development.

Fabio Di Silvestro is a software engineer currently working as a back-end developer. From 2020 to 2022, Fabio worked in the AVENTRA Integration Team at ALSTOM. He graduated with Master (Double) Degree in Computer Science with Specialisation in Software Engineering in 2020 at university of L’Aquila and Mälardalen university. Fabio has experience in IT and Software Engineering and currently his development focuses on a custom Object Oriented framework for test automation.

Inderjeet Singh is a software integration and simulation manager at ALSTOM. He has a master degree in software engineering from Mälardalen university. He has extensive working experience in embedded development and control systems domain in industry. In addition, he is the TCMS Alstom research coordinator.

Mehrdad Saadatmand is a senior researcher in software engineering at RISE Research Institutes of Sweden in the city of Västerås where he also leads the Software Testing Group. Mehrdad holds a Ph.D degree from Mälardalen University in Sweden. He has extensive experience of working in various collaborative industrial research projects, particularly several large-scale European ones.
Henry Muccini is a full professor in computer science at the University of L'Aquila, Italy. He got his PhD degree in computer science from the University of Rome, La Sapienza. Henry leads research on machine learning for software architecture adaptation, legacy systems to microservice migration, software architecture descriptions, and ICT for the cultural heritage. Henry is coordinating the Smart City and Communities Lab at the University of L'Aquila.

How to cite this article: Bucaioni A, Di Silvestro F, Singh I, Saadatmand M, Muccini H. Model-based generation of test scripts across product variants: An experience report from the railway industry. J Softw Evol Proc. 2022;34(11):e2498. doi:10.1002/smr.2498