


An Industrial Case Study for Adopting Software Product Lines in Automotive Industry

An Evolution-based Approach for Software Product Lines (EVOA-SPL)

Karam Ignaim João M. Fernandes 
Department of Informatics / ALGORITMI Centre
Universidade do Minho
Braga, Portugal

ABSTRACT

Software Product Lines (SPLs) seek to achieve gains in productivity and time to market. Many companies in several domains are constantly adopting SPLs. Dealing with SPLs begin after companies find themselves with successful variants of a product in a particular domain. The adoption of an SPL-based approach in the automotive industry may provide a significant return on investment. To switch to an SPL-based approach, practitioners lack a reengineering approach that supports SPL migration and evolution in a systematic fashion.

This paper presents a practical evolution-based approach to migrate and evolve a set of variants of a given product into an SPL and describes a case study from the automotive domain. The case study considers the need to handle the classical sensor variants family (CSVF) at Bosch Company.

Using this study, we performed a contributed step toward future switch of the CSVF into the SPL. We investigated the applicability of the proposed evolution-based approach with a real variants family (using the textual requirements of the CSVF) and we evaluated our approach using several data collection methods. The results reveal that our approach can be suitable for the automotive domain in the case study.

CCS CONCEPTS

• Software and its engineering → Software development techniques → Reusability

KEYWORDS

Software Product Line, Feature Model, case study, variability.

ACM Reference format:

Karam Ignaim and João M. Fernandes. 2019. An Industrial Case Study for Adopting Software Product Line in Automotive Industry: An Evolution-based Approach for Software Product Lines (EVOA-SPL). In *Proceedings of ACM 23rd International Systems and Software Product Line Conference (SPLC 2019)*. Paris, France, 8 pages.
<https://doi.org/10.1145/3307630.3342409>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

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

© 2019 Association for Computing Machinery. ACM ISBN 978-1-4503-6668-7/19/09...\$15.00. <https://doi.org/10.1145/3307630.3342409>

1 Introduction

Some companies in the market need to handle multiple variants that have some characteristics in common [1]. One of the key factors to improve productivity and consequently to reduce the cost to handle multiple variants is software reuse. In fact, many variants in a specific industrial domain have been implemented by reusing pieces of existing variants rather than building them from scratch [2]. An SPL can be seen as a family of variants that have been developed with explicit concern about commonality and variability during development process [3].

SPLs have been used successfully by industry to promote reuse. For example, several reports from large companies such as Bosch, Nokia, and Philips observe benefits with their use, especially with respect to the reduction in time to market [4]. Because of long term dealing with variants family, companies need to handle them in a systematic way. Therefore, SPLs can be a suitable option in this case. To switch to SPLs, companies need to adopt an approach that supports migration of variants family into an SPL and evolution of SPLs [5]. Regarding SPLs, a prerequisite for systematic reuse is to explicitly specify evolutionary change of variants family in a variability model [6]. Feature models (FMs) constitute a suitable to model address migration and evolution of SPLs [3]. Depending on the abstraction level, features may refer to a prominent or distinctive user-visible characteristic or functionality of variants family [7, 8].

Actually, SPLs have been commercially applied in many industry domains [9]. Related to the challenge of adopting SPLs taking care of their evolution in the automotive industry, initially migration of automotive variants family into an SPL often starts with an analysis of variability [5]. The automotive industry faces challenges to manage variability among variants family, due to its product domain. Numerous research papers point out the necessity of requirements and system modelling to manage variability in the automotive industry [10]. Most proposals in this context do not consider textual requirements, even though they enable to manage variability from the beginning [11][12].

This work proposes an approach to address the use of SPLs at the requirements level. The purpose of this paper is to recommend a practical evolution-based approach that supports a reengineering process to migrate automotive variants family into an SPL and to evolve an SPL after it has been established. This research work applies the proposed approach to the CSVF at Bosch Company and analyses the approach through a case study involving the CSVF and the classical sensor development team (CSDT). The approach includes different activities for both the reverse and

forward engineering chaining phases and it provides guidelines for the CSDT in an SPL context for the automotive industry. The focus of this work remains at the requirements-level.

2 Related work

Despite the research works that address adoption of SPLs [2, 13], there is still a lack of the approaches that propose guidelines or methods for performing SPLs migration and evolution in a systematic way. However, most of the presented approaches were evaluated using toy examples or open-source applications (e.g., ArgoUML) [14, 15]. Few approaches were evaluated using industrial case studies [16, 17].

In addition, other works propose refactoring patterns and notations that fit the SPLs context. Moreover, they are applicable to FMs [18, 19]. As a common practice, the authors in [18, 19] evaluate their work in mobile applications and health information domains respectively. Another work [20] proposes a generic framework for managing collections of related cloned products into an SPL, where the products are refactored into a single-copy SPL. Empirically, this work analyses three industrial case studies of different organizations.

A systematic study provides an overview of current research on reengineering of existing systems into SPLs [21]. This study concludes that reengineering of existing systems into SPLs is an active research topic with real benefits in practice. Moreover, it motivates new research in the adoption of systematic reuse in software companies. In this context, it reports the lack of sophisticated refactoring, the need for new metrics and measures, and more robust evaluation.

A model-driven approach to support software engineers in handling source code variability of software variants in the automotive domain is proposed in [22]. This work contrasts with ours, since it does not consider the requirements of the automotive domain, when modelling and managing variability [22].

Close to our work, the authors in [23] introduce a systematic reuse method called Variation Point Method (VPM), which models variability in a process that starts with common requirements. Important research works related to managing variability are [4, 8, 24-28].

However, not only is our approach designed to consider the variants that are related to the same family in the automotive domain but also it is used to address commonality and variability of variants family at a higher level of abstraction using variability model (i.e., an FM). Moreover, it is planned to use SPLs refactoring. In addition, our approach is analysed and evaluated in an industrial case study.

3 Our approach

Based on the general process of evolution of SPLs and the analysis of our industrial case study from the automotive domain, we propose EVOA-SPL, an evolution-based approach for SPLs. EVOA-SPL supports the evolution of an SPL focusing on migration of the variants into an SPL. To tackle this challenge, the approach considers variability at the requirements-level. EVOA-SPL supports a reengineering process towards systematic reuse and consistent evolution of an SPL.

As shown in Figure 1, EVOA-SPL adopts a reengineering process, which consists of two distinct phases: the reverse engineering phase and the forward engineering phase. Each of them is summarily described in the following subsections. The FM works as a common model that is shared between two phases. Phase 1 derives an FM and phase 2 upgrades and refines it. The adoption of a reengineering process is amenable for EVOA-SPL, since it considers existing variants and it is incremental (support a new change to the requirements in a systematic way).

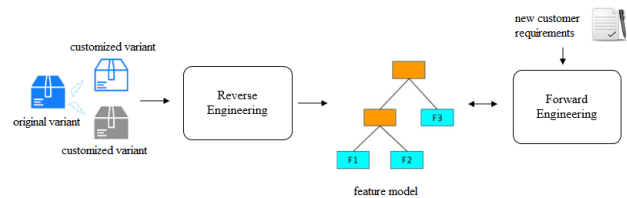


Figure 1: The EVOA-SPL approach phases.

3.1 The reverse engineering phase

The reverse engineering phase consists of three main activities: (1) the difference analysis, (2) the variability analysis, and (3) the feature model synthesis. The main input for this phase are the textual requirements of each two variants. Thus, this phase identifies commonality and variability among automotive variants family at the requirements-level. It uses the textual requirements of only two variants and derives an FM. The project manager and domain expert may be consulted to clarify any information about the variants; to select proper textual requirements from the document, and to revise and confirm the derived FM.

3.2 The forward engineering phase

The forward engineering phase consists of three main activities: (1) the bootstrapping, (2) the evolution, and (3) the FM refactoring. The reverse engineering phase delivers an FM that is used as an input for this phase. The activities of this phase use the current version of the FM and the textual requirements of another / new variant to evolve the SPL.

The bootstrapping activity imports variants family (one by one) according to the decisions of the domain experts and the project manager. Once an SPL has been bootstrapped, the evolution process can start. It evolves an SPL with a new variant upon receiving a new customer request. Both activities derive and store features of a (new) variant into a features list (FL) and refactor (i.e., refine) the current FM with the requirements (features) of a (new) variant that is not supported yet by the SPL.

3.3 EVOA-SPL activities

In order to migrate the variants into an SPL and then support an SPL evolution, EVOA-SPL goes through a reengineering process. EVOA-SPL uses the textual requirements of automotive variants, which were initially created with ad-hoc approaches. Before performing the EVOA-SPL activities, one is required to study the domain-specific issues related to automotive variants family, like notations, technique, or process steps [29]. The EVOA-SPL activities that the software engineers can follow to migrate automotive variants family into an SPL and to support its evolution are the following (see Figure 2).

3.3.1 *Activity 1.* The **difference analysis** activity captures and identifies similarities and differences between two variants. This activity (i) writes textual requirements of each variant into atomic requirements (ARs), (ii) gives each AR a unique id, and (iii) stores ARs in the so-called requirements document (RD). Furthermore, (iv) it uses a proper text-based comparison tool to specify common and optional ARs among several RDs. The comparison concerns changes between RDs in terms of matched, added, deleted, and modified ARs. Figure 3 depicts the RD structure at the end of this activity.

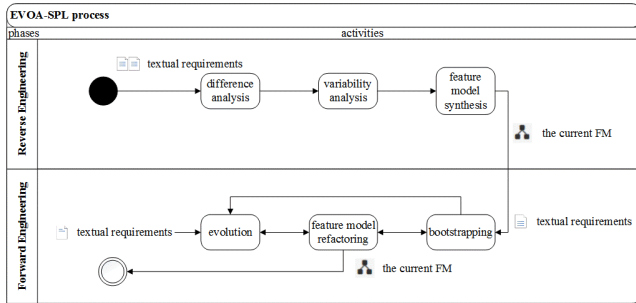


Figure 2: EVOA-SPL activities.

3.3.2 *Activity 2.* The **variability analysis** activity performs further processing and identifies commonality and variability among variants. The main inputs of this activity are RDs of two variants. The variability analysis activity (i) specifies common and optional ARs from RDs of two variants and (ii) stores those classified ARs in a single master document (SMD). It represents an initial adequate view of commonality and variability among automotive variants family, since variant members are related to the same family and shared the same domain architecture.

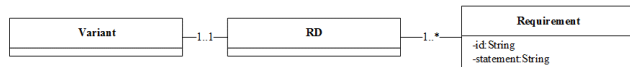


Figure 3: The structure of RD.

Additionally, this activity (iii) uses the “common”, “optional”, “OR-Group”, and “XOR-Group” keywords to classify the variability-pattern of each AR in an SMD (please see Table 1). The variability analysis activity (iv) uses text-parsing to extract the set of keywords of each AR in an SMD that have important meaning to identify features and their dependencies. It parses actions (verbs), objects including instruments, technologies, services, parameters (attributes) and signal names. These keywords set helps to identify feature names and their dependencies. This activity (v) suggests a proper name for the feature using the keywords set and based on the suggestions from the domain experts. Next, this activity identifies features and their variability-pattern (please see Table 1) using the feature identification method (see Section 3.4). Finally, this activity (vi) transfers and organizes variability information into an FL, which contains a set of features, each one with a predefined variability-pattern and a feature-relationship. A feature-relationship can have two types; parent-child relationship and dependency relationship.

Moreover, in this list, a feature is related to a set of ARs that are responsible for its specification in an SMD.

Table 1: The feature variability-pattern.

Common	The feature must be included in every variant.
Optional	The feature may be included (or not) in every variant, but it is not necessary or required
OR-Group	The feature is part of a group of features and if their parent is included, at least one of those features is included in the variant.
XOR-Group	The feature is part of a group of features, and if their parent is included exactly one of those features is included in the variant.

3.3.3 *Activity 3.* The **feature model synthesis** activity synthesizes an FM. This activity maps variability information from an FL to an FM and delivers a model that contains: feature name, variability-pattern, and feature-relationship. One of the relationships that needs to be captured is a parent-child relationship to show that one feature is a child of another one. The other type of relationship is dependency relationship; requires dependency means “feature A requires feature B” to be selected and excludes means “feature C requires feature D” to not be selected. In other words, both features C and D should never be simultaneously used in the same variant.

To build an FM, this activity (i) reads features from an FL sequentially. Then it (ii) uses mapping table to transform from an FL into an FM (please see Table 2). The mapping table relates variability notations between an FL and an FM. Finally, this activity (iii) builds the FM with a given modelling tool (e.g., “FeatureIDE”) [12].

To build the FM, the software engineers (i) draws features inside the respective symbol (rectangle), (ii) defines variability-pattern for each feature, and (iii) draws child-feature under their parent-feature and defines feature-relationship among them.

Table 2: The mapping notations between an FL and an FM.

feature list	feature model	
title		root feature
feature		Feature
common feature		mandatory feature
optional feature		optional feature
OR-Group		Or
XOR-Group		alternatives
parent-child relationship		parent-child relationship

3.3.4 *Activity 4.* The **bootstrapping** activity adds remaining members of variants family (one by one) to contribute to an SPL. This activity (i) uses textual requirements of a given variant, (ii) derives features of that variant, and (iii) stores them in an FL. Finally, this activity (iv) evolves current FM to encompass features of the variant using an *FM refactoring scenario* (see feature model refactoring activity).

3.3.5 *Activity 5.* The **feature model refactoring** activity refactors and refines the current FM with features of a (new) variant that is not supported yet by the SPL. This activity uses a catalogue of sound FM refactorings to perform transformations that improve and increase the current FM.

These refactorings appear in [18]. Fortunately, these refactorings are reasoned and proved in [18]. The feature model refactoring activity uses an *FM refactoring scenario* that works as follow. It (i) reads a feature from an FL (top to down) and (ii) matches feature (and its variability-pattern) with nodes (and their variability-pattern) of current FM. Based on the match-status, the scenario (iii) performs the proper an FM refactorings. An *FM refactoring scenario* considers the following match-status:

1. Status 1: a feature requires changes in current FM (please see Table 3). This leads to apply to the current FM the proper refactoring that appear in [18].
2. Status 2: a feature appears in the current FM with the same variability information. This leads to keep the current FM unchanged. Thus, the refactoring are only applied in case of differences between an FL and an FM.
3. Status 3: a feature appears as a common feature in the current FM and does not appear in an FL of a (new) variant. This leads to apply the proper refactoring that appear in [18] to the current FM, in order to transform the common feature into an optional feature.

An *FM refactoring scenario* repeats until reaching the final feature in an FL. It is worth to mention that if evolution is a feature deletion, an *FM refactoring scenario* of the EVOA-SPL approach does not support this case. Since it needs the requirement engineer to check and approve this evolution. In addition, it requires not only to delete feature from current FM but also to remove the associated components and feature-software-unit.

Table 3: The feature changes in the current FM.

1.	Add a new common feature.
2.	Add a new optional feature.
3.	Add a new Or (OR-Group) feature.
4.	Add a new alternative (XOR-Group) feature.
5.	Transform an optional feature into a common feature.

3.3.6 *Activity 6*. The **evolution** activity propagates requirements (features) of a new variant to the current FM. Actually; this activity evolves an SPL to encompass a new variant once it has been bootstrapped. This activity (i) uses textual requirements of a new variant, (ii) derives features for this variant, and (iii) stores them in an FL. Finally, this activity (iv) evolves the current FM to encompass features of a new variant using an *FM refactoring scenario* (see feature model refactoring activity).

3.4 The feature identification method

We proposed the Feature Identification Method (FIM), which consists of three main interconnected parts. FIM identifies features and their variability-pattern and feature-relationship in an FL. The main parts of FIM, which is inspired by forward chaining in expert system [30], are documented below.

Part1. FIM variability knowledge (VK) is a collection of facts (see below) that are applied to each ARs sequentially.

1. A set of ARs represents a feature.
2. A set of common ARs represents a common feature.
3. A set of variable ARs represents an optional feature.
4. Grouped ARs belongs to the same main feature.

5. A main AR forms a parent-feature.
6. Grouped ARs form child-feature.
7. AR forms a child-feature when it has one value and does not carry options or alternatives.
8. AR forms OR-Group or XOR-Group when it is within a group of options. The former is used when at least one or more ARs (options) can be included in a variant and the latter is used when just one AR (option) can be included in a variant. This can be observed using an SMD.
9. AR that required another AR to be included forms a dependent relationship.
10. AR that required another AR not to be included forms an independent relationship.

Part 2. FIM variability rules (VRs) are a collection of rules (see below) that have an if-then statement format. The if-then statement consists of two sides, on the left-hand (LHS) is if-side and on the right-hand (RHS) is then-side. We can apply the rule on AR whenever a given AR matches the LHS of VR.

1. R1: If AR is a child-feature then it has a parent-child relationship with parent feature.
2. R2: If AR forms OR-Group then it has OR-Group with parent-feature.
3. R3: If AR forms XOR-Group then it has XOR-Group with parent-feature.
4. R4: If AR forms a dependent relationship with other AR then it has requires relationship.
5. R5: If AR forms an independent relationship with other AR then it has excludes relationship.

Part 3. The FIM process works on ARs of an SMD and updates them sequentially. The FIM starts the first iteration with the VK and applies them sequentially on each AR until the last AR in the SMD is reached. In case that one of the VK is not satisfied, (does not match an AR) it will be skipped. After ARs are updated by the VK, the FIM starts the second iteration with VRs. The method searches ARs until one of them matches LHS of the VRs and then applies the RHS of this VR on that AR. The FIM stops when reaches last AR in an SMD. Now an SMD is updated with variability information that makes it rich enough to feed feature model synthesis activity.

4 The case study

In order to evaluate the EVOA-SPL approach, we conducted a case study following the guidelines, which are presented in [31]. According to [31], the case study is composed of four major process steps to be walked through: planning, design, data collection and, analysis and reporting.

4.1 Planning

Good planning is necessary for the success of the case study. Therefore, we planned several issues.

Objective. The objective of the case study is to evaluate the EVOA-SPL approach in the automotive domain using the CSVF. The case study will be conducted with the CSDT who has previous experience in the automotive domain development.

Treatment. Our case study has one treatment, which is the EVOA-SPL approach.

Objects. The object of our case study is the CSVF, which are implemented based on AUTOSAR architecture. We used four variants that are related to the CSVF, specifically, the textual requirement of each variant.

Subjects. The subjects of the study are the CSDT and the project manager.

Methods. We planned to perform our case study in two stages. In the first stage, we took the role of a software engineer and we applied the EVOA-SPL approach on the CSVF. During the execution of the EVOA-SPL approach process, initially, we derived the SMD and the FM. After that, we started to bootstrap the CSVF in the SPL and then we evolved the SPL with the new variant using an *FM refactoring scenario*.

In the second stage, we planned to evaluate the EVOA-SPL approach and the generated artefacts using several data collection methods, where we have found the empirical study including survey, interview, and observation are applicable to the environment of our case study

4.2 The variants family used in the study

The CSVF has been developed and customized by the team for more than 3 years, to satisfy the needs of different customers in the automotive domain. Moreover, they have around 20 major releases. The CSVF has source code written in C and includes 50 packages. A number of features have been added and modified, while the variants have been evolved over time. To conduct this case study, we used the CSVF, which has a fixed architecture (AUTOSAR).

The CSVF consists of three variants, which were developed according to the needs of customers in the automotive domain, and a new variant, which was an upcoming variant scheduled for being developed in the near future upon receiving a new customer request. All the variants were cloned from the original variant (platform variant, which is often evolved from the platform developed and successfully used by the first customer) and then modified according to customer needs. To simplify the following discussion, we designate for every variant in the CSVF a number and we called the upcoming variant a new variant.

The subjects involved in the study had full access to the variants family documentation and the code. Even though we took the role of software engineers, we applied the EVOA-SPL approach on CSVF artefacts; we had limited access to the documentation and code. For confidentiality, no all details of the CSVF are provided.

4.3 Data Collection

Case Study Environment. The study was conducted in the software development department, during October 2017- June 2018 at Bosch Company.

Procedure. The study was conducted in two stages.

Stage 1. We took the role of a software engineer and we applied the EVOA-SPL approach on the CSVF using the textual requirement (i.e., requirement specification document) of each variant.

Stage 2. We evaluated the effectiveness and the efficiency of the EVOA-SPL approach from the point view of CSDT and we evaluate the correctness of the EVOA-SPL approach concerning project manager perspective.

Summary of Generated Artefacts. Following the EVOA-SPL approach, the main activities during the execution of the case study in stage 1 were difference analysis, variability analysis, feature model synthesis, bootstrapping, feature model refactoring, and evolution.

Firstly, the SMD of variants family was defined, capturing commonality and variability between variant 1 and variant 2. Secondly, the FM was derived, representing the SPL at a high level of abstraction. The model presents common features of the SPL, which are the *Message* including the *Transmit* and the *Receive* messages, the *Diagnosis*, the *Monitoring* and the optional features, which are the *Calculations* and the *Interface support*. On the other hand, the *Transmit* feature has two features involving the *Layout* and the *Signals*, where the *Layout* feature has two alternatives, which are *Layout 1* and *Layout 2*. This means that the variants have two different message layouts, one for each variant: the first layout is *Layout 1*, which has five bits; the second one is *Layout 2*, which has seven bits.

Concerning the *Signals* feature, it can be contained different signals for the variants (variant 1 and variant 2). Some of these features are common (which are *Signal 1*, *Signal 2*, and *Signal 3*), some of them are optional (which are *Signal 4* and *Signal 5*), and some of them are within alternative group (which are *Flag 1*, *Flag 2*, *Algorithm 1*, and *Algorithm 2*).

After structuring the FM, it is the time to bootstrap the CSVF into the SPL and then evolve the SPL. The former evolves the current FM with the requirements (the features) of the remaining variant of variants family (variant 3). The later evolves the current FM with the requirements (the features) of a new variant.

Firstly, the CSVF must be bootstrapped into the SPL. For that, the features of variant 3 are identified and stored in the FL. At this point, the FM is refined with the features of variant 3 using an *FM refactoring scenario*. Now the SPL is bootstrapped completely and the commonality and the variability of the CSVF are both presented by the current FM.

The *Signal 6* feature, which exists in the FL of variant 3 and does not exist in the current FM, presents new features. *Signal 6* feature appears as a new feature in the list, since it does not exist in the current FM. At the same time, this feature represents a change at the SPL level (adding a new feature). As shown in Figure 4, this change is propagated to the SPL and the new feature (*Signal 6*) is moved from the FL of variant 3 to the FM using an *FM refactoring scenario* (the scenario uses Refactoring 12 “add optional node” [18]).

Once the bootstrapping is completed, the SPL can be evolved with features of the new variant, using an *FM refactoring scenario*. Prior to this step, the features of the new variant are derived and stored in the FL of the new variant. The FL presents the features of the new variant, the *Identification* feature appears as a new type of the *Message* feature and *Flag 3* feature appears as another alternative of the *Flag* feature, since they do not exist in the current FM. These new features represent changes in the SPL level (adding new features).

As shown in Figure 4, these changes are propagated to the SPL and the new features (*Identification* and *Flag 3*) are moved from the FL of new variant to the FM using an *FM refactoring scenario*. Regarding the *Identification* feature, the scenario uses Refactoring 12 “add optional node” and regarding *Flag 3* feature, the scenario uses Refactoring 5 “add new alternative” [18].

The last step is to perform a typical documentation and readable artefacts preparation. In the end, the approach activities have been applied to the CSVF and the FM has been derived, synthesised and evolved, as shown in Figure 4, where the refactoring locations are highlighted. At this point, stage 2 of the case study can be started to assess the EVOA-SPL approach by CSDT and the project manager.

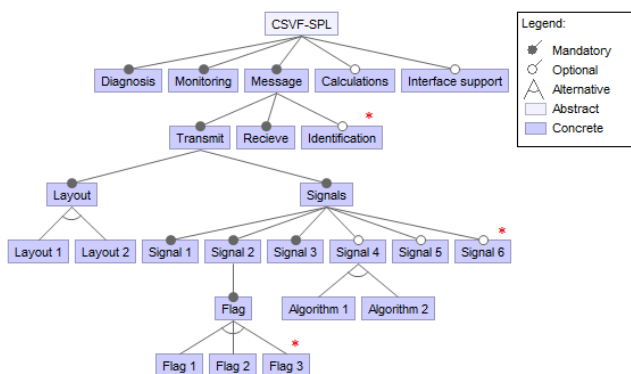


Figure 4: The current FM of the CSVF.

The objective of stage 2 of the case study is to assess the effectiveness and the efficiency of EVOA-SPL according to the CSDT and its correctness according to the project manager perspective. Hence, we formulated the following hypotheses to measure the effectiveness, efficiency, and correctness of the EVOA-SPL approach.

- H1:** EVOA-SPL is effective.
- H2:** EVOA-SPL is efficient.
- H3:** EVOA-SPL is correct.

4.4 Execution of Data Collection

We prepared many documents that suited to the collecting data methods. We designed a presentation that introduces the EVOA-SPL activities and the generated artefacts to the members of CSDT. In addition, we prepared a survey that is a part of an empirical study. The survey includes a questionnaire with 22 questions. These questions were formulated by using a combination of descriptive, behaviour, and attitudinal questions. The answers were given using ordinal and nominal scale response formats. One of the most important members of the team is undoubtedly the project manager. Thus, it is of special importance to dedicate her a survey, which contains measurement items related directly to the hypotheses.

For the empirical study, we defined a set of items to be evaluated by asking the CSDT members to perform specific tasks and then answer questions while using directly with the EVOA-SPL

approach. Ideally, to compare the developer’s solutions with a possibly-correct solution, in order to investigate our hypotheses, we have defined “correct” solutions for the tasks. The aim was to evaluate the effectiveness and the efficiency of the CSDT while applying EVOA- approach. For that, we defined five dependent variables.

Regarding effectiveness, we defined: **Effectiveness-SMD**, which is calculated as the ratio between the number of correct variability retrieval scenarios from the SMD that the CSDT identified and the total number of correct retrievals. **Effectiveness-FM** is calculated as the ratio between the number of correct feature retrieval scenarios from the current FM that the CSDT member identified and the total number of correct retrievals. **Effectiveness-EVO** is calculated as the ratio between the number of correct evolution scenarios to the current FM that the CSDT member performed and the total number of correct evolutions.

Regarding efficiency, we defined **Efficiency-SMD** as the ratio between the number of correct variability retrieval scenarios from the SMD that the CSDT member identified and the total time he spent. **Efficiency-FM** is computed as the ratio between the number of correct feature retrievals scenario from the current FM that the CSDT member identified and the total time she spent. **Efficiency-EVO** is calculated as the ratio between the number of correct evolutions to the current FM that the CSDT member performed and the total time he spent.

Regarding correctness, **Correctness-EVOA-SPL** is defined as the ratio between the number of positive feedback from the project manager about the validity of the variability information provided by EVOA-SPL and the number of survey questioners that is written and dedicated to get the feedback of project manager.

For the direct methods to collect data, we prepared an interview with the CSDT members to get direct feedback related to EVOA-SPL by establishing open questions. Moreover, in order to get a deeper understanding, we used the observation. We informed the project manager that we are going to investigate the use of EVOA-SPL by the CSDT members in their normal daily work. We agreed with them to use EVOA-SPL during two weeks (daily for one hour). To perform a data collection, the following steps were conducted:

1. We met the software developers of the CSDT (the project manager and seven developers). We started the first session by training including the presentation about EVOA-SPL and training exercises on using its capabilities.
2. We established the second session; we took four days with eight sessions, each session of one hour. We met the software developers individually (it is not recommended to simultaneously pause the work of many developers in an industrial company for a long period).
3. We performed the empirical case study. In the first 10 minutes, we took the developer background information using a form, and then we gave them 3 tasks¹ to perform

¹ According to the policy of the company, we hide the tasks.

using EVOA-SPL. Each task consists of 3 tags². Finally, we asked the developer to answer the survey about EVOA-SPL.

Data collection through semi-structured interviews was performed using questions about a set of subjects related with EVOA-SPL. The interview dialog was guided by a set of questions. Simultaneously, we observed the CSDT members while using EVOA-SPL and we took notes about those observations..

4.5 Analysis and reporting

We performed a qualitative analysis related to the dependent variables to prove our three hypotheses H1-H3. The qualitative analysis was undertaken based on the collected data that was performed earlier. Related to the tasks that were performed by the developers within the empirical study. Table 4 summarises the results of the qualitative analysis.

Firstly, we investigate the effectiveness of task 1 (**Effectiveness-SMD**), which is related to the task of retrieving the variability information from the SMD. We compared the developer's solutions with the correct solutions. The result reveals that the developers were able to achieve a high percentage; they solved around 93% of the total tags related to this task. In what concerns the effectiveness in task 2 (**Effectiveness-FM**) of using the current FM to retrieve variability information, the CSDT members were able to retrieve correctly around 89% of the total variability information. The effectiveness in task 3 (**Effectiveness-EVO**) to perform evolution scenarios to the current FM, the CSDT members were able to perform correctly around 85% of the total evolutions.

We repeated the analysis for the same tasks, but we measured the time used and we estimated the efficiency. The results show that the CSDT members took around 10 minutes to complete task 1, which originates an **Efficiency-SMD** value of 0.26. The CSDT members took around 12 minutes to complete task 2, so *Efficiency-FM* is equal to 0.21. Finally, the CSDT members took around 15 minutes to complete task 3, with an **Efficiency-EVO** value of 0.18.

Table 4: Mean and max for the data analysis of dependent variables.

subjective dependent variable	mean	max
Effectiveness-SMD	0.93	1.00
Effectiveness-FM	0.89	
Effectiveness-EVO	0.85	
Efficiency-SMD	0.26	0.30
Efficiency-FM	0.21	0.25
Efficiency-EVO	0.18	0.20
Correctness-EVOA-SPL	0.86	1.00

We repeated the analysis for the same tasks to estimate the efficiency but this time we asked the CSDT to perform the tasks using the normal approach³ (and its related artefacts), which is adopted by the company for a long time to develop the CSVF to satisfy customer's needs. The results show that the CSDT members took around 20 minutes to complete task 1, with an

Efficiency-SMD value of 0.14. The CSDT members took around 30 minutes to complete task 2, with an **Efficiency-FM** value of 0.08. Finally, the CSDT members took around 28 minutes to complete task 3, with an **Efficiency-EVO** value of 0.09. Figure 5 shows a chart that compares the EVOA-SPL approach with the normal approach regarding efficiency while the CSDT performing the tasks related to the empirical study. In total, the comparison reveals that the CSDT performed the tasks related to the empirical case study more efficiently than using the normal approach.

Regarding the correctness, we depend on the feedback of the project manager. The project manager highly agreed that the results of the EVOA-SPL approach are valid (the **Correctness-EVOA-SPL** is equal to 86%). Due to the manual observation while the CSDT members were performing exercises related to EVOA-SPL, the members were able to understand and interact with the approach smoothly. They believe that the activities and the artefacts of the EVOA-SPL approach can be used to achieve the intended objectives. Moreover, the survey results reflect high satisfaction, which helped us to find a new hypothesis related to our approach (it is useful).

During the interviews, the project manager and (most of) the CSDT members confirmed that EVOA-SPL supports the move towards an SPL. Moreover, they confirmed that the approach helped to retrieve features that took many hours to search for them in the artefacts of the CSVF. Concretely, we analysed the developer's feedback that was provided using the survey questionnaire. The developers not only agreed that EVOA-SPL can be used to manage variability over the CSVF, but they also agreed that it can help to reduce the effort.

4.6 Threats to Validity

The main threat to the validity of the empirical study is the missing in the quantitative analysis. For that, we plan to perform this analysis in our future work. Moreover, the design of the survey questionnaire, the number of developers who shared in the empirical study, and the exchange of information between the developers are other main reasons that may threaten internal validity.

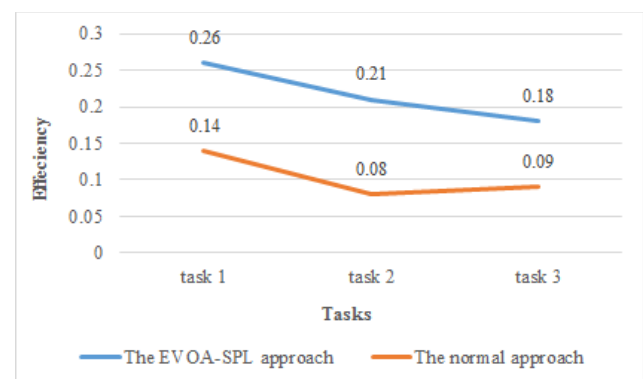


Figure 5: The comparison of the efficiency between the EVOA-SPL approach and the normal approach.

5 Conclusions

In this paper, we introduce the EVOA-SPL approach to support migration of automotive variants into an SPL and then support the

² Tag is a request for the developer to retrieve information or to make change using the EVOA-SPL approach.

³ For confidentiality, many details of the normal approach are not provided

evolution of the SPL, focusing on the textual requirements of the variants. We present a case study in the automotive domain conducted for the classical sensor variants family at Bosch Company, using the guidelines described in [31]. The results of the case study have shown that EVOA-SPL can be suitable for the automotive industry.

As future work, we need to perform additional empirical evaluation with larger and more complex SPLs. We want also to improve/enrich the approach by reducing the number of activities and the number of steps within each activity. Moreover, we plan to build a tool to automate the manual steps of EVOA-SPL.

ACKNOWLEDGMENTS

The University of Minho and Bosch Company supported this research. We thank our colleagues from the classical sensor development team at Bosch Company. Especially André L. Ferreira and Jana Seidel for their active collaboration and support. Special acknowledgment to the spirit of Helder Boas, who passed away after he offered the help and support to this research work.

REFERENCES

- [1] Vander Alves, Nan Niu, Carina Alves, and George Valença (2010). Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, 52(8), 806-820.
- [2] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon (2015). Bottom-up adoption of software product lines: a generic and extensible approach. In *Proceedings of the 19th Int. Conf. on Software Product Line*, 101-110.
- [3] Ra'fat Al-Msie'Deen, Abdelhak Seriai, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, and Hamzeh Eyal Salman (2013). Feature location in a collection of software product variants using formal concept analysis. In *Int. Conf. on Software Reuse*, 302-307.
- [4] Fernando Wanderley, Denis Silva da Silveira, João Araujo, and Maria Lencastre (2012). Generating feature model from creative requirements using model driven design. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*, 18-25.
- [5] Andreas Metzger and Klaus Pohl (2014). Software product line engineering and variability management: achievements and challenges. In *Proceedings of the Future of Software Engineering*, 70-84.
- [6] Klaus Pohl and Andreas Metzger (2006). Variability management in software product line engineering. In *Proceedings of the 28th Int. Conf. on Software Engineering*, 1049-1050.
- [7] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake (2016). Feature-oriented software product lines. Springer-Verlag.
- [8] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans (2013). Feature model extraction from large collections of informal product descriptions. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, 290-300.
- [9] Mikael Svahnberg and Jan Bosch (1999). Evolution in software product lines: Two cases. *Journal of Software Maintenance: Research and Practice*, 11(6), 391-422.
- [10] Olivier Renault (2014). Reuse/Variability Management and System Engineering. In *Poster Workshop of the Complex Systems Design & Management Conference CSD&M 2014*, 173.
- [11] Yang Li, Sandro Schulze, and Gunter Saake (2017). Reverse engineering variability from natural language documents: A systematic literature review. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*, 133-142.
- [12] Daniela Rabiser, Paul Grünbacher, Herbert Prähofer, and Florian Angerer (2016). A prototype-based approach for managing clones in clone-and-own product lines. In *Proceedings of the 20th International Systems and Software Product Line Conference*, 35-44.
- [13] Samuel A. Ajila and Patrick J. Tierney (2002). The FOOM method-modeling software product lines in industrial settings. In *Proceedings of the 2002 Int. Conf. on Software Engineering Research and Practice*.
- [14] Marcus Vinicius Couto, Marco Tulio Valente, and Eduardo Figueiredo. (2011). Extracting software product lines: A case study using conditional compilation. In *15th European Conference on Software Maintenance and Reengineering*, 191-200.
- [15] Jabier Martinez, Tewfik Ziadi, Tegawende F. Bissyande, Jacques Klein, and Yves Le Traon (2015). Automating the extraction of model-based software product lines from model variants. In *30th IEEE/ACM Int. Conf. on Automated Software Engineering*, 396-406.
- [16] Vander Alves, Pedro Matos, Leonardo Cole, Paulo Borba, and Geber Ramalho (2005). Extracting and evolving mobile games product lines. In *Int. Conf. on Software Product Lines*, 70-81.
- [17] Bo Zhang and Martin Becker (2012). Code-based variability model extraction for software product line improvement. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*, 91-98.
- [18] Vander Alves, Rohit Gheyi, Tiago Massoni, Uirá Kulesza, Paulo Borba, and Carlos Lucena (2006). Refactoring product lines. In *Proceedings of the 5th Int. Conf. on Generative Programming and Component Engineering*, 201-210.
- [19] Mohammad Tanhaei, Jafar Habibi, and Seyed-Hassan Mirian-Hosseinebadi (2016). A feature model based framework for refactoring software product line architecture. *Journal of Computer Science and Technology* 31(5), 951-986.
- [20] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik (2013). Managing cloned variants: a framework and experience. In *Proceedings of the 17th International Software Product Line Conference*, 101-110.
- [21] Wesley KG Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed (2017). Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering*, 22(6), 2972-3016.
- [22] Cem Mengi, Christian Fuß, Ruben Zimmermann, and Ismet Aktas (2009). Model-driven Support for Source Code Variability in Automotive Software Engineering. In *1st MAPLE Workshop*, 44-50.
- [23] Diana L. Webber and Hassan Gomaa (2004). Modeling variability in software product lines with the variation point model. *Science of Computer Programming* 53(3), 305-331.
- [24] Hitesh Yadav and A. Charan Kumari (2018). Analysis of Features using Feature Model in Software Product Line: A Case Study. *International Journal of Education and Management Engineering* 8(2), 48-57.
- [25] Andreas Metzger, Klaus Pohl, Patrick Heymans, Pierre-Yves Schobbens, and Germain Saval (2007). Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *15th IEEE Int. Requirements Engineering Conference (RE 2007)*, 243-253.
- [26] Jabier Martinez, Tewfik Ziadi, Jacques Klein, and Yves Le Traon (2014). Identifying and visualising commonality and variability in model variants. In *European Conference on Modelling Foundations and Applications*, 117-131.
- [27] Nili Itzik, Iris Reinhartz-Berger, and Yair Wand (2015). Variability analysis of requirements: Considering behavioral differences and reflecting stakeholders' perspectives. *IEEE Transactions on Software Engineering*, 42(7), 687-706.
- [28] Steven She, Uwe Ryssele, Nele Andersen, Andrzej Wąsowski, and Krzysztof Czarnecki (2014). Efficient synthesis of feature models. *Information and Software Technology* 56(9), 1122-1143.
- [29] Matthias Weber and Joachim Weisbrod. 2002. Requirements engineering in automotive development-experiences and challenges. In *Proceedings IEEE Joint International Conference on Requirements Engineering*. IEEE, 331-340.
- [30] Ashwini Rupnawar, Ashwini Jagdale, and Samiksha Navsuppe (2016). Study on Forward Chaining and Reverse Chaining in Expert System. *Int. Journal of Advanced Engineering Research and Science*, 3(12), 60-62.
- [31] Johan Linaker, Sardar Muhammad Sulaman, Martin Höst, and Rafael Maiani de Mello (2015). Guidelines for Conducting Surveys in Software Engineering v. 1.1.