An Empirical Study of Performance Using Clone & Own and Software Product Lines in an Industrial Context

Jorge Echeverría[a,*], Francisca Pérez[a], José Ignacio Panach[b], Carlos Cetina[a]

[a] *SVIT Research Group, Universidad San Jorge, Zaragoza, Spain*
{jecheverria,mfperez,ccetina}@usj.es
[b] *Universitat de València, Avenida de la Universidad, s/n, 46100 Burjassot, Valencia, Spain*
joigpana@uv.es

Abstract

Context: Clone and Own (CaO) is a widespread approach to generate new software products from existing software products by adding small changes. The Software Product Line (SPL) approach addresses the development of families of products with similar features, moving away from the production of isolated products. Despite the popularity of both approaches, no experiment has yet compared them directly.

Objective: The goal of this paper is to know the different performances of software engineers in the software products development process using two different approaches (SPL and CaO).

Method: We conducted an experiment in the induction hobs software environment with software engineers. This experiment is a single factor experiment where the factor is the approach that is used to develop software products, with two treatments: (SPL or CaO). We compared the results obtained by the software engineers when they develop software products related to effectiveness, efficiency, and satisfaction.

Results: The findings show that: 1) the SPL approach is more efficient even though the number of checking actions required by this approach is greater than the number required by the CaO approach; 2) the SPL approach offers more possibilities than software engineers need to perform their daily tasks; and 3) software engineers require better search capabilities in the CaO approach. The possible explanations for these results are presented in the paper.

Conclusions: The results show that there are significant differences in effectiveness, efficiency, and satisfaction, with the SPL approach yielding the best results.

Keywords:
Clone and Own, Software Product Line, Experiment, Empirical evaluation

1. Introduction

In industrial contexts where software development is based on the generation of new software from legacy software, two approaches have been used successfully: Software Product Line (SPL) and Clone and Own (CaO). The CaO approach is based on the generation of software product families from legacy software products with small modifications [31]. On the other hand, the SPL approach is based on the systematic reuse of a set of software components to derive new software products. These new products are created from existing elements instead of developing software from scratch [24].

In empirical studies related to the SPL and CaO approaches, we observe two characteristics: 1) most studies are conducted with students that play the role of subjects; and 2) to our knowledge, there are no previous papers that have conducted empirical experiments in industry with the aim of comparing the SPL and CaO approaches. Thus, there is a need to conduct empirical studies with software engineers to compare SPL and CaO approaches from an industrial perspective in the real world. In order to bridge this gap, the main contribution of this paper is the design and conduction of an experiment to compare the SPL and CaO approaches in industry.

CaO and SPL are two approaches used for software development in industrial environments. In some contexts, depending on the circumstances both approaches

[*]Corresponding author. Tel.: +34 976060100 mail: jecheverria@usj.es

are used. In the industrial context where the experiment was conducted the predefined approach is SPL but CaO is used when the time to develop a software product is short, to perform noncommercial products (e.g., prototypes), and if the software engineers think that the feature will not be reused in the future. In these circumstances, the software engineers avoid formalizing the features as opposed to the SPL approach that requires the feature formalization. Then, it becomes especially relevant how is the software engineers' performance in these contexts.

The goal of this paper is to determine which approach is better: SPL or CaO. This paper proposes an experiment design that is based on a single factor experiment using a between-subjects design. The factor is the type of approach while the response variables are three metrics used to measure quality according to ISO 25000: effectiveness (the percentage of task performed correctly), efficiency (the ratio between effectiveness and time spent to perform the tasks), and satisfaction (the subjective feelings of the software engineers after finishing the experimental tasks) [1]. The experimental problem consists of four tasks where the subjects have to develop a software product using one approach. The tasks, the software sets,and the language used are the same for both approaches, this ensures that treatments are given in the same context. The experimental subjects were software engineers in the induction hob division of our industrial partner company, the BSH group. BSH is the largest manufacturer of home appliances in Europe and one of the leading companies in the sector worldwide (the brand portfolio is composed of Bosch and Siemens, among others). Their induction division has been producing induction hobs for the last 15 years.

The results show that there are significant differences for these three variables: effectiveness, efficiency, and satisfaction have better values with the SPL approach than with the CaO approach. Our results show findings that are relevant for improving the software development process with the CaO and SPL approaches. Related to the CaO approach the software engineers asked about possibilities of improvement in the search capabilities in the CaO approach. On the other hand, regarding the SPL approach is more efficient than CaO approach but the software engineers spend more time checking actions.

The paper is structured as follows: Section 2 analyzes other works that have shown empirical evaluations on SPL and CaO. Section 3 explains the background of the SPL and CaO approaches. Section 4 describes the design of the experiment. Section 5 shows the statistical results. Section 6 discusses the results. Section 7 deals with the threats to validity. Finally, Section 8 presents some relevant conclusions.

2. Related Work

Since there are so many concepts to be evaluated in a SPL, there is an important number of works that have dealt with studies in this field. Next, we describe the search strategy that was used to answer the **research question**: *What evaluations have been done in the field of SPLs and Clone and Own?* The **search string** used is *("software product lines" AND "experiment") OR ("software product lines" AND "empirical evaluation") OR ("software product lines" AND "assess") OR Clone and Own.* The **inclusion criterion** is: (IC1) *Works that evaluate SPLs or CaO through empirical experiments, case studies, or theoretically.* The **exclusion criteria** are: (EC1) *Works that do not describe in detail the goal, metrics, and outcomes of the evaluation*; (EC2) *Works without any validation.* The search was done in May, 2019 using Scopus. This tool searches journals and conference proceedings from a broad set of libraries.

The primary works selected from the set of papers retrieved by the search have been classified based on the topic of the evaluation: requirements elicitation, evolution, complexity of metrics, reliability, comparisons, Software Product Line, and Clone and Own. Below, we describe the papers found in each one of these categories. A summary of all of the primary works is shown in Table 1. For each work, we summarize the type of evaluation (empirical experiment, case study, or theoretical work), the number and type of subjects, the goal of the evaluation, the variables used in the evaluation, and the outcomes.

There are several works that focus on evaluating the process of requirements elicitation in SPL. Adam and Schmid [2] have conducted an empirical experiment with 26 students who play the role of subjects. The goal of the experiment is to analyze two elicitation approaches regarding effectiveness during requirements elicitation. The results show that the elicitation technique proposed by the own authors is more effective than a traditional one. Bonifacio el at. [7] have also conducted an empirical experiment in the field of requirements elicitation in SPLs. The results have been extracted from a family of three experiments with 12, 24, and 16 subjects, respectively, where all of them are students. The goal of the experiment is to compare two techniques to specify use case scenarios for SPLs. The comparison is performed through the variable *ef-*

Author	Topic	Type	Subjects	Goal	Variables	Result
Adam and Schmid [2]	RE	Experiment	26 students	Compare two techniques of requirements elicitation for SPL	Elicitation effectiveness	The proposed method is more effective than traditional methods
Bonifacio et al. [7]	RE	Experiment	52 students in three replications	Compare two techniques to specify SPL requirements	Effort	The most modular technique obtain better results
Bagheri and Gasevic [5]	EV	Experiment and theoretical work	15 students	Evaluate a set of structural feature model metrics	External quality attributes	Some metrics are considered better than others as indicators for maintainability
Figueiredo et al. [18]	EV	Case Study	The authors	Assess the capabilities of aspects in SPL	Modularity, change propagation, feature dependency	Aspects provide more stable designs
Michalik et al. [33]	EV	Experiment	17 professionals	Analyze the evolution with minimal interruption of services	Correctness, logistic system availability, confidence level	The architecture-centric approach improves the correctness
Tizzei et al. [40]	EV	Theoretical work	The authors	Evaluate the impact of components in the evolution of SPL	Number of modules and number of operations	The combination of components and aspects is the best option
Dubinsky et al. [13]	EV	Case Study	The authors	To investigate the code clonning culture.	Perceived advantages and disadvantages	Issues that prevent the adoption of clonning
Fenske et al. [17]	EV	Case Study	The authors	To propose an approach to migrate multiple cloned product variants into an SPL.	Effectiveness	An incremental migration process from CaO to SPL.
Berger and Sturm [37]	CM	Experiment	116 students	Evaluate the comprehensibility of SPL specified with UML	Comprehension questions	Providing explicit reuse guidance improves comprehensibility
Marcolino et al. [32]	CM	Experiment	35 students	Evaluate the complexity of SPL architectures	Complexity metrics	Composed metrics for complexity are the most reliable
Vale et al. [42]	CM	Case study	The authors	Compare methods to derive thresholds	Complexity metrics	A set of recommendation to define metrics thresholds
Koziolek et al. [28]	R	Case study	The authors	Identify commonalities and variabilities in domains of SPLs	Domain metrics	A set of recommendations to analyze domains
Krishnan et al. [29]	R	Case study	The own authors	Investigate failures in components of SPLs	Number of failures per component	Common components present the least failures
Constantino et al. [11]	CO	Experiment	84 students	Compare two SPL tools	Easy of use, strengths and weaknesses	The main issues are in interfaces and lack of user guides
Dermeval et al. [12]	CO	Experiment	5 students	Compare two modeling techniques for specifying SPLs	Time to change, impact of changes, correctness	OWL individuals require less time and are more flexible than OWL classes
González-Huerta et al. [22]	SPL	Experiment	92 students	Validate a MDD method for building SPL	Effectiveness, efficiency, perceived ease of use, perceived usefulness and intention to use	Better architectures are obtained though the MDD method, enhancing subjects' satisfaction
Guana and Correal [23]	SPL	Experiment	2 subjects	Validate a MDD method to automate the definition of SPLs through reusable components	Test cases	The MDD approach improves reusability but involves more time
Santos and Kulesza [38]	CaO	Case Study	The own authors	Analyze the complexity to integrate merge conflicts of a cloned web system	Number of merging conflicts	There are many semantic conflicts and it is feasible to use merge analysis to integrate tasks.
Schlie et al. [39]	CaO	Case Study	8 domain experts	Propose and evaluate a procedure to assist model engineers in maintaining and evolving existing variants	Performance precision	A technique to support engineers in maintaining and reusing existent models.
Fisher et al. [19]	CaO	Case Study	The authors	An approach to enhance CaO in the development of software product variants	Quality of the composition, guidance	An approach with the benefits of CaO and systematic reuse.
Ghabach et al. [21]	CaO	Case Study	The authors	An approach to support the derivation of new product variants using CaO	Configuration scenarios, number of products, number of assets and cost	The approach can save time and effort during product derivation.
Krüger et al. [30]	CaO	Case Study	The authors	Support developers to identify common features in cloned systems	Lines of code, differences between the features, dependencies between features	The process is suitable to identify features and present commonalities and variability in cloned systems.
This paper	CaO SPL	Experiment	10 software engineers	Know the performance when assets are reused to develop software products	Effectiveness, efficiency, satisfaction	The process is suitable to identify features and present commonalities and variability in cloned systems.

Topic RE: Requirements elicitation, EV: Evolution, CM: Complexity of Metrics, R: Reliability, CO: Comparisons, SPL: Software Product Line, CaO: Clone and Own.

Table 1: Related Work

fort. The results show that most modular technique reduces the effort required by the analysts.

The evolution of SPL is a feature that must be considered in any architecture. There are several works that have analyzed this evolution. One of these works was performed by Bagheri and Gasevic [5], who have conducted an experiment with 15 students to evaluate a set of structural feature model metrics both theoretically and empirically. The evaluation has been done measuring external quality attributes. The results define a subset of metrics that have been identified as correlating with maintainability. Figueiredo et al. [18] have reported a case study to analyze the evolution of two SPLs in order to asses the capabilities of aspects in SPLs. The variables analyzed are modularity, change propagation, and feature dependency. The results show that the use of aspects provides a more stable design. Moreover, aspects scale well for dependencies that do not involve shared code. Michalik et al. [33] have conducted another experiment based on evolution. In this case, the subjects were composed of 17 professionals, with the goal of analyzing the evolution of updates in SPLs with minimal interruptions in services. The variables analyzed are the correctness of an update, the availability of the logistic system, and the confidence level. The results show that the architecture-centering approach improves the correctness of updates and reduces the interruption of services during updates. Tizzei et al. [40] have done a theoretical study to evaluate the impact of positive and negative changes in components and aspects on the evolution of SPLs. The study focuses on analyzing the number of modules and the number of operations in different SPL systems. The results show that the combination of aspects and components obtains the best values. Dubinsky et al. [13] addressed the lack of empirical knowledge about the software development practices of companies that use cloning to implement product lines. They conducted an empirical study to investigate the cloning culture in six industrial software product lines realized via code cloning. They presented a set of recommendations to efficiently develop and manage software product line assets. Fenske et al. [17] proposed a process to migrate cloned product variants to a feature-oriented SPL. They evaluated their approach on five cloned product variants. Their approach reduced synchronization effort compared to CaO development and thus reduced the long-term costs for maintenance and evolution.

Other empirical research focuses on analyzing the comprehensibility and complexity of SPLs. Berger and Sturm [37] have conducted an experiment with 116 students to study the comprehensibility of domain models of SPL systems through UML. The evaluation was done using comprehension questions regarding reuse guidance and variability specification. The results show that providing explicit reuse guidance has the greatest influence on the results. Moreover, the variability specification also improved comprehensibility. Marcolino et al. [32] have conducted an experiment to evaluate the complexity of SPL architectures. Thirty-five students played the role of analysts participating in the experiment. This experiment is a replication of another baseline experiment. In this case, the subjects were less-qualified than in the baseline. The metrics used in this replication were two different metrics to measure complexity. The results show that even less-qualified subjects obtain better results for complexity when metrics are composed. Vale et al. [42] have also performed an analysis of complexity in SPL. The research is a case study to compare three methods to derive thresholds for metrics that measure complexity in SPLs. As a result of that work, there is a list of recommendations and good practices to define thresholds.

There are other works that evaluate features of SPLs such as potential or reliability. In this group of works, we find the work of Koziolek et al. [28], who have performed a case study to identify commonalities and variabilities among SPLs to identify the potential of each product. The study uses several metrics to analyze the domain of SPLs and, as an outcome of this analysis, there is a set of recommendations to analyze the domains of SPLs. Krishnan et al. [29] have done a case study to analyze the occurrence of severe failures in four Eclipse releases. The case study was done using the number of failures as metrics. The results show that fewer failures occur in components that implement common functionalities.

There are also empirical studies that focus the analysis on comparing different techniques for working with SPLs. Constantino et al. [11] have compared two SPL tools using 84 students that play the role of subjects. This comparison was based on the ease of use of each tool and their strengths and weaknesses. The results show that the main issues observed in both tools are related to interfaces and a lack of examples and tutorials. Dermeval et al. [12] have compared two alternative approaches on ontology-based feature modeling (OWL classes versus OWL individuals). The comparison was done with five students, measuring the time to perform changes, the structural impact of changes and the correctness of the changes. The results show that using OWL individuals requires less time to change and is more flexible than using OWL classes.

A few of the existing analyses have been focused on the context of Model-Driven Development (MDD). One of these works was done by González-Huerta et al. [22], who carried out a family of four experiments using 92 students as subjects with the goal of evaluating a MDD method to develop SPL. The metrics used for the evaluation are effectiveness, efficiency, perceived ease of use, perceived usefulness, and intention to use. The results show that the architecture defined using the method based on MDD is the best. Moreover, the subjects that worked with MDD think that it is easier to use, more useful, and more likely to be used. Guana and Correal [23] focus on evaluating a MDD method to automate the definition process of a SPL through the use of reusable components. Two subjects participated in the experiment and the metrics were based on a suite of tests. The results show that the use of MDD improves the reusability of components, even though it increases the time required to learn the tool.

There are not many experiments on the topic of CaO in the literature. In this field, Santos and Kulesza [38] have done an exploratory case study that analyzes the complexity of integrating existing merge conflicts of a cloned large-scale web system. The study is based on the number of conflicts that appear in merging actions. The results show that there is a predominance of semantic conflicts in merge actions and that it is feasible to use merge analysis approaches to integrate tasks from one clone to another. Schlie et al. [39] proposed an advanced comparison procedure, the Matching Window Technique, to improve the software development process with CaO. The authors conducted three case studies with real-world models from the automotive domain. In these case studies, they studied performance and precision to validate their technique. In [19], Fisher et al. proposed an approach to enhance CaO in the development and maintenance of software product variants. They evaluated their approach in six diverse case studies of different sizes and domains. The results show that their approach leverages the benefits of CaO while still providing the benefits of systematic reuse. Ghabach et al. [21] proposed an approach to support the derivation of new product variants based on CaO by providing the possible scenarios in terms of operations to be performed in order to accomplish the derivation. They validated their approach in a case study where the results showed that the provided support can reduce the amount of time and effort that are required to achieve a product derivation. In [30], Krüger et al. proposed a process to identify and map features among legacy systems, suggesting a visualization approach. The authors assessed the process in a case study. Their findings indicate that the process is suitable to identify features and present commonalities and variability in cloned systems.

After analyzing all of the primary works, we can draw some important conclusions. First, there is a lack of evaluations conducted in industry. All of the works except for Michalik et al. [33] and Schlie et al. [39] used students, or the own authors of the works themselves played the role of subjects. Second, to our knowledge, there is no a previous comparative experiment of SPL versus CaO. Third, the use of metrics such as effectiveness, efficiency, and satisfaction seems to be consolidated in empirical experiments. Therefore, in this paper, we describe an empirical evaluation conducted in industry using effectiveness, efficiency, and satisfaction to compare the SPLs and the CaO approaches. According to existing works, this new contribution is a clear step forward in evaluating SPLs in a real context of use, bridging a gap in previous works.

3. Background

This section presents the two approaches used in the experiment (CaO and SPL).

3.1. Clone and Own

CaO is an approach for generating new software products in a family of software products [13]. This approach consists of reusing software artefacts from existing software products and adding small changes to generate a new software product with new characteristics [31]. Figure 1 shows an example of a family of software products generated with CaO approach [19]. Product P1 consists of two features (BASE and LINE). After some time, another product (Product P2) is generated from a variant of two features (BASE and LINE) from Product P1 (using the CaO approach), so Product P2 holds two CAO relationships with a previous product, Product P1. Moreover, Product P2 comprehends a new feature (COLOR), which has been created from scratch. After, another product (Product P3) is built with a new feature (RECT), a variant of three features (BASE, LINE, and COLOR) from Product P2. Hence, Product P3 holds three CaO relationships with Product P2, one for each reused feature. In total, this family of products comprises 3 products, 4 features, and 5 CaO relationships [36].

The CaO approach has traditionally been used to generate software products in industrial environments. This approach has been applied when the software engineers must develop a new software product that is very similar to a previously existing one [13]. The CaO approach might be efficient depending on certain circumstances: catalogue of products, products complexity,

Products	BASE	LINE	RECT	COLOR
Product P1	X	X		
Product P2	X	X		X
Product P3	X	X	X	X

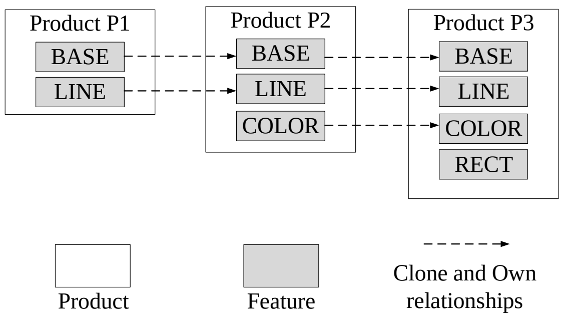


Figure 1: Family of software products generated with CaO approach.

and the development organization and its software engineering practices [4].

In industrial scenarios, the CaO approach is carried out manually and relies on the knowledge that software engineers have of the software models, the code, and the families of software products. In that context, the difficulty of managing of software products increases: software products with long and complex implementations arise, and the maintenance of software products over long periods of time by different developers decreases the control over them. In these scenarios, the engineers tasked with new product developments often lack knowledge of the entirety of the products and their implementation details [31]. On the other hand, the maintenance of many independent products leads to multiple problems like inefficient feature updates or bug fixing, duplicate functionality, redundant and inadequate testing, etc [13].

In contrast, in certain scenarios, the CaO approach facilitates the generation of new software products, the traceability of these products, and helps maintain a homogeneous development style among the different software products in a family [31].

3.2. Software Product Lines

A SPL is "*a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission*" [10].

The possible software products of a SPL may be described using feature models. Figure 2, adapted from [6], shows the feature models of a SPL. The relationships show in Figure 2, according to proposal described in [26], are the following [14]:

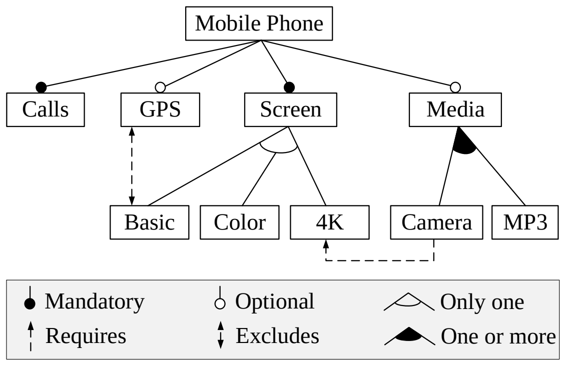


Figure 2: Feature model of a SPL.

- *Mandatory*: a child feature has a mandatory relationship with its parent feature when it is required to appear in a given product whenever its parent feature appears in that product.

- *Optional*: a child feature has an optional relationship with its parent feature when it can appear or not in a given product whenever its parent feature appears in that product.

- *Or–relationship*: a set of child features have an or–relationship with their parent feature when one or more child features can be selected in a given product when the parent feature appears in that product.

- *Alternative*: a set of child features have an alternative relationship with their parent feature when only one of them must be selected in a given product when their parent feature appears in that product.

- *Requires, Excludes*: a cross–tree relationship like A requires B means that in any product where feature A appears, feature B must also appear. On the other hand, a relationship like A excludes B means that both features cannot appear in the same product at the same time.

A SPL is an approach for the strategic and systematic reuse of a set of assets within an organization. A SPL is composed of a product line architecture, a set of software components, and a set of derivative products. The main concept in a SPL approach addresses the development of families of products with similar features, moving away from the production of isolated products. Variability and commonalities for generating new software products have been explored in order to optimize the

SPL	Number of Products	Number of Features
ZipMe	32	7
VOD	32	11
GameOfLife	65	15
ArgoUML	256	13
ModelAnalyzer	5	5
SPL used in this paper	46	81

Table 2: Examples of SPL.

software development based on product families. These products are built using a core asset base instead of being developed one by one from scratch [24].

The main advantage of a SPL is the systematic reuse of the common infrastructure which is shared to create different product variants [4]. Other benefits of a SPL approach to develop software are: improved productivity, increased software quality, decreased cost, decreased labor needs, and decreased time to market [16]. Table 2 shows the characteristics of 6 SPLs, the table includes the SPL used in this paper [19].

On the other hand, the use of a SPL approach can generate some difficulties: the incorrect definition of portfolio, the guarantee of the quality of maintenance due to the explosion of dependencies, identification of the products affected by a change, and the implementation of the same change in several versions of the products [43].

Our goal is to analyze the software engineers' performance with two approaches (CaO and SPL) in an industrial environment. For this reason, we conducted an experiment at the BSH induction hob division. Both approaches (CaO and SPL) have a set of 46 induction hob models, corresponding to products that are currently being sold or that will be launched to the market in the immediate future. Both approaches use a Domain Specific Language for induction hobs named Induction Hobs Domain Specific Language (IHDSL) [20]. With regard to the products complexity, each of the induction hob models is composed of more than 500 elements, including around 100 class elements on average.

4. Experiment Design

4.1. Objective

The goal of our research is to know the different software engineers' performance when they use legacy software products in the product development process using different approaches (SPL and CaO). Following Wohlin et al.'s guidelines [44], the goal of our study is to:

Analyze the engineers' performance when they use legacy software products to develop software products;

For the purpose of bridging in the gap of in empirical evaluation of this topic;

With respect to the different approaches used (SPL and CaO);

From the viewpoint of software engineers;

In the context of the BSH induction hob division.

In relation to the above goal, we seek both to compare the software engineers performance with the CaO and SPL approaches from the point of view of quality. According to ISO 25000, quality in use can be measured through effectiveness, efficiency, and satisfaction [1]. This definition of quality leads to define the following research questions:

RQ1 Are there differences between the SPL approach and the CaO approach regarding effectiveness in developing software products?

RQ2 Are there differences between the SPL approach and the CaO approach regarding efficiency in developing software products?

RQ3 Is the subjective satisfaction of software engineers in developing software products different when they use the SPL approach or the CaO approach?

Taking these Research Questions into account, we have formulated the following null hypotheses to answer them:

- $\mathbf{H_{01}}$: There is no difference in the effectiveness of the SPL approach and the CaO approach.
- $\mathbf{H_{02}}$: There is no difference in the efficiency of the SPL approach and the CaO approach.
- $\mathbf{H_{03}}$: There is no difference in the subjective satisfaction of software engineers using the SPL approach and the CaO approach.

4.2. Participants

The subjects were ten software engineers who work for the induction hob division in BSH. These engineers are experts in developing software. They spent from 1 to 12 years working as software engineers (a mean of 5.4 years). Five subjects that work daily with CaO performed the experimental problem with the SPL approach, while five other subjects that work daily with SPL used the CaO approach. Therefore, the subjects had never developed software with the approaches used before this experiment. This lack of experience was solved by the subjects using a tutorial and training in the approach applied (CaO or SPL). On the other hand, all the subjects develop daily software with both the Domain Specific Language and the software sets used in the experiment. Before conducting the experiment, the experimenters trained the subjects, who applied tasks

	User	Gender	Age	Education Level	Job	Time in Job (years)
SPL	user SPL1	M	33	D	E	12
	user SPL2	M	45	D	E	5
	user SPL3	M	31	D	E	2
	user SPL4	M	30	D	E	5
	user SPL5	M	35	D	E	10
CaO	userCaO1	F	31	D	E	4
	userCaO2	F	32	D	E	10
	userCaO3	M	24	D	E	1
	userCaO4	M	27	D	E	2
	userCaO5	M	28	D	E	3

Table 3: Users. *D:Degree, E: Software engineer.*

that were similar to the ones used in the experiment (create and modify products). The tools used in the experiment had been customized to develop software products in the induction hob division of the company. An explanation about the tool used for the SPL approach is shown in the link: https://youtu.be/nS2sybEv6j0. The predefined approach is SPL but CaO is used in some circumstances like to perform noncommercial products or when the available time to develop a software product is short.

Besides the subjects, an instructor and an observer were also involved. The instructor provided the information about the approaches used, gave tutorials, participated in the training, interviewed the subjects after the experiment, clarified doubts, and designed the correction. The instructor was not involved in the writing of this paper. The observer took notes, recorded the interviews for further analysis, and corrected the tasks.

4.3. Defining Variables

4.3.1. Factor and Block Variables

The single factor in the experiment is the type of approach used to perform four tasks in the context of an induction hob company (SPL and CaO). These approaches are explained in section 3. The tools used to operationalize both treatments are real tools that are used daily in the company.

The experimental design is a between-subjects design with one factor and two treatments. Each subject only uses one treatment of the factor. We have the experimental task as a block variable since we are analyzing the response variables for each task independently of the others. This design allows us to better analyze what is happening in each treatment per task even though we are not really interested in looking for differences among tasks. This is the reason why we opted for blocking the task variable.

4.3.2. Response Variables and Their Metrics

The aim of this experiment is to determine the effectiveness, efficiency, and satisfaction of software engineers when using the SPL approach VS the CaO approach. The response variables are defined as follows:

- Effectiveness is defined as the percentage of experimental tasks performed correctly by the engineer without assistance. Commercial products documentation was used to define a task as correct. The developments of these products had been documented and this documentation was used as ground truth. The task is decomposed into a set of subtasks in such a way so we can measure whether or not each subtask was completed successfully. This way, the effectiveness of the task is calculated as the aggregation of the effectiveness of its subtasks. Since there are subtasks with different levels of difficulty, the aggregation of the effectiveness per task was done through weights. The weight of each subtask is the time that we estimated it would take to be performed. Therefore, subtasks that require more estimated time are considered to have more weight. We used the Keystroke-Level Model [27] to assign an estimated time to each subtask. The aggregation to calculate the effectiveness of the task consists in adding the estimated time of the subtask only if the subtask was done successfully. When all of the subtasks are added (without errors in any subtask), the time added means 100 % effectiveness for the task. When at least one subtask was not done successfully, the effectiveness of the task is calculated as the percentage of successfully done subtasks taking to account the estimated times. Table 4 shows how task T1 (with the SPL approach) is decomposed according to the Keystroke-Level Model method: estimated times for subtasks 1.1, 1.2, 1.3, and 1.4 are respectively 6.3, 3.7, 6.2, and 4,9. For example, if we have a sample where subtask 1.1.1 (estimated time of 1.2) was not done successfully, the aggregation for Task 1 would be 19.9, which means an efficiency of 94.31 % (100*19.9/21,1).

- Efficiency is the ratio between the effectiveness and the time spent (in minutes) to perform the task according to the Common Industry Format (CIF) for Usability Test Reports [3].

- Satisfaction is measured using a satisfaction questionnaire filled out by the engineers after finishing the experimental tasks using each approach (SPL

1. Task: T1		**Time**
1.1	**Sub-task: Select the induction hob**	
1.1.1	Initiate the task (decide to do)	1.2 s
1.1.2	Remember the induction hob reference	1.2 s
1.1.3	Find the induction hob	1.2 s
1.1.4	Point to induction hob	1.1 s
1.1.5	Double click on induction hob	0.4 s
1.1.6	Notice the selected induction hob in the editing window	1.2 s
1.2	**Sub-task: Select the module**	
1.2.1	Remember the module reference	1.2 s
1.2.2	Find the module	1.2 s
1.2.3	Point to module	1.1 s
1.2.4	Click on module	0.2 s
1.3	**Sub-task: Replace the module**	
1.3.1	Remember the new module reference	1.2 s
1.3.2	Find the new module	1.2 s
1.3.3	Point to new module	1.1 s
1.3.4	Click with the button on the right on the new module	0.2 s
1.3.5	Find the option *replace*	1.2 s
1.3.6	Point to option *replace*	1.1 s
1.3.7	Click on option *replace*	0.2 s
1.4	**Sub-task: Apply only to one induction hob**	
1.4.1	See the dialog box	1.2 s
1.4.2	Determine the right choice	1.2 s
1.4.3	Find the right choice	1.2 s
1.4.4	Point to chosen button	1.1 s
1.4.5	Click on chosen button	0.2 s

Table 4: Detailed for task T1 decomposition (SPL approach)

and CaO). The questionnaire was composed of ten questions using a Likert scale [41].

Note that we have analyzed one measure for effectiveness and one measure for efficiency per experimental task. This way, for each subject, we have as many measures for effectiveness and efficiency as tasks in our experimental problem. In the case the satisfaction metric, we have only one measure per subject since we have only one questionnaire to be completed at the end of the experiment.

4.4. Instruments

4.4.1. Demographic Questionnaire

This includes questions to identify the profile of each subject. The information asked for in the questionnaire is the following: their educational level, length of time working in the current department (years), age, gender, knowledge about developing software with integrated development environments (only subjects using the SPL approach), knowledge about developing software using the CaO approach (only subjects using the CaO approach) and experience with tools for automatic generation of code.

4.4.2. Task sheet

The experimental tasks were chosen for importance and frequency of use. In the context of software product development from existing software products, the following tasks that compose the experimental problem were stated

- **T1**: To develop a new software product. The development of a new software product from existing ones is depicted in ❶ of Figure 3. With the CaO approach, the new software product (New P) is performed using existing source code. With the SPL approach, the new software product (New P) is performed using existing features.

- **T2**: To develop a new software product with a new feature (SPL approach) or new source code (CaO approach). This new feature or new source code is created from an existing one. Figure 3 [❷ and ❸] with the CaO approach depicts the creation of new source code from the existing one and then a new software product is performed with this new source code. Figure 3 [❷ and ❸] with the SPL approach depicts the creation of a new feature from an existing one and then a new software product is performed with this new feature.

- **T3**: To modify an existing software product. The modification of a software product is depicted in ❹ of Figure 3.

- **T4**: To modify an existing software product with a new feature (SPL approach) or new source code (CaO approach). This new feature or new software artefact is created from an existing one. Figure 3 [❺ and ❻] with the CaO approach depicts the creation of new source code from existing one and then a software product is modified. Figure 3 [❺ and ❻] with the SPL approach depicts the creation of a new feature from an existing one and then a software product is modified with this new feature.

In the context of our experiment, we have instantiated these tasks to the context of induction hobs. We have defined four tasks that have a similar level of difficulty. The level of difficulty in a task depending on the amount of software products to create or modify, the amount the artifacts software used to solve the task, and the amount of subtasks in the task. The four tasks are the following:

- T1_IH: A new induction hob must be developed from the induction hob IH011. This induction hob contains the module MOD006. This module must

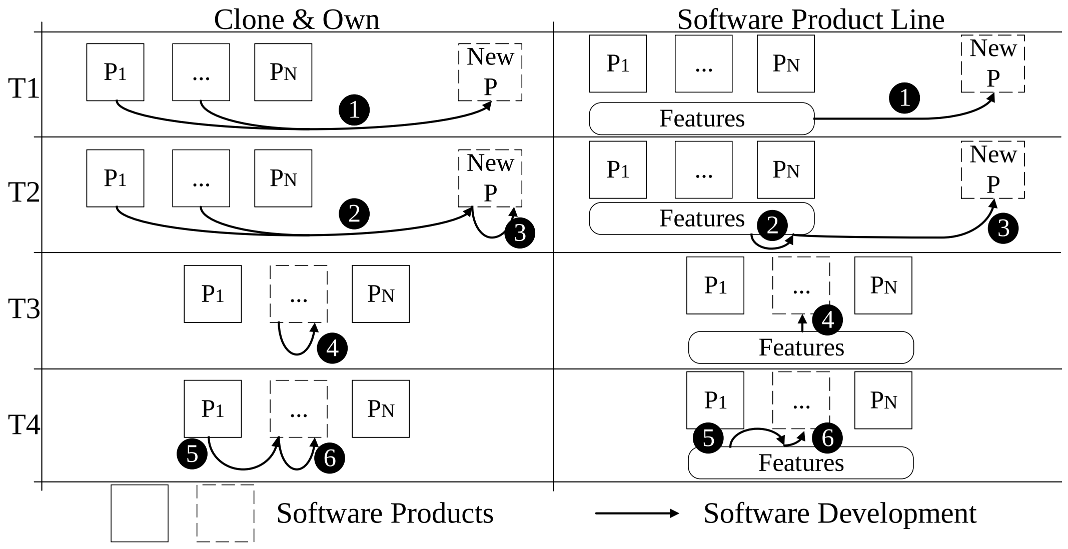


Figure 3: Tasks of the Experimental Problem

be replaced with the module MOD014 to create a new induction hob.

- T2_IH: A new module must be created from module MOD020 to place in a new induction hob. The value of the parameter VMAX must be changed to 39 in the module MOD020 to create a new module named MOD020B. Then, the new module MOD020B replaces the module MOD020 in the induction hob IH005 to create a new induction hob.
- T3_IH: The inverter INV014958 of the module MOD016 must be changed. The inverter must be replaced by the inverter INV015231. The induction hob IH017 contains the module MOD016.
- T4_IH: The value of the parameter VMAX that is set in the module MOD019 of the induction hob IH051 must be changed. A new module MOD019B must be created from the module MOD019 where the value of the parameter VMAX has to be changed to 44. The module MOD019 must be replaced by the new module MOD019B in the induction hob IH051.

These tasks were written on the task sheet given to the subjects. They had to develop the software products according to the four tasks.

4.4.3. Recordings

The recordings contain the subjects' performance when they developed the products according to the above tasks and the subjects' claims about the two approaches used in the experiment. These recordings allowed us to calculate of the time spent to perform the tasks and also to determinate where the subjects made mistakes and the sources of these mistakes.

4.4.4. Satisfaction Questionnaire

This questionnaire is the System Usability Scale (SUS), which determines the subject's subjective satisfaction with the approach used. Measuring user satisfaction provides a subjective usability metric. The questionnaire is composed of ten questions using a Likert scale. In the original SUS, the word *system* is replaced by *approach*. The SUS yields reliable results with only ten questions [41]. The SUS questions address different aspects of the subject's reaction to the approach used as a whole (e.g., "I found the approach unnecessarily complex", "I felt very confident using the approach") as opposed to asking the users to assess specific features of the system (e.g., visual appearance, organization of information, etc.). The data collected with the SUS must be put on a spreadsheet in order to be processed to obtain the value of SUS [8].

4.4.5. Interview

The objectives of this interview were: 1) to determine the understanding by the subject about the approach used; and 2) to obtain qualitative data from the subject's comments. The interview is composed of both open questions and closed questions. The aim of the closed questions is to check the understanding of the approach by the subjects. The aim of the open questions is to detect the concepts or process of the approach used that are more problematic for subjects, along with the real causes of the problems [25]. For instance, one question is "Do you know if you have created unnecessary assets?".

The materials resulting from carrying out the experiment can be found at https://svit.usj.es/CaO_vs_SPL/.

4.5. Experimental Procedure

The experiment was conducted at BSH in Zaragoza and took one hour per subject. The procedure [see Figure 4] was the following for both approaches:

1. The subjects were given information about the goals and objectives of the experiment. They were told that it was not a test of their abilities. They were also informed that their interaction would be recorded.
2. The subjects attended a small tutorial about the approach to be used. This tutorial was taught by the instructor. The average time spent on this tutorial was four minutes since the tasks used in the experiment were very short. For example, Task 1 is usually done in two minutes.
3. The subjects were asked to fill in a demographic questionnaire prior to testing.
4. The subjects were then given a series of clear instructions that were specific for the process of product development according to the task statements. They were advised to try to accomplish the tasks without any assistance and that they should only ask for help if they felt unable to complete the task on their own.
5. The subjects were asked to create four software products with reusable assets according to the four tasks detailed in subsection 4.4.2. These development processes were used to calculate effectiveness and efficiency. To avoid a possible ceiling effect, there was no time limit to complete the product development.
6. The observer wrote down if the subtasks were done successfully. This information was complemented with a video recording of the session in order to understand the pros and cons of each approach.
7. After finishing the four tasks of the experimental problem, the subjects were then asked to complete the SUS questionnaire. This questionnaire was used to calculate the satisfaction of the approach used.
8. The subjects were interviewed by the instructor about the tasks that they performed.
9. The observer reviewed the recordings of the subjects performing the tasks in order to calculate the efficiency, effectiveness and satisfaction. Finally, the interviews about the tasks were transcribed.

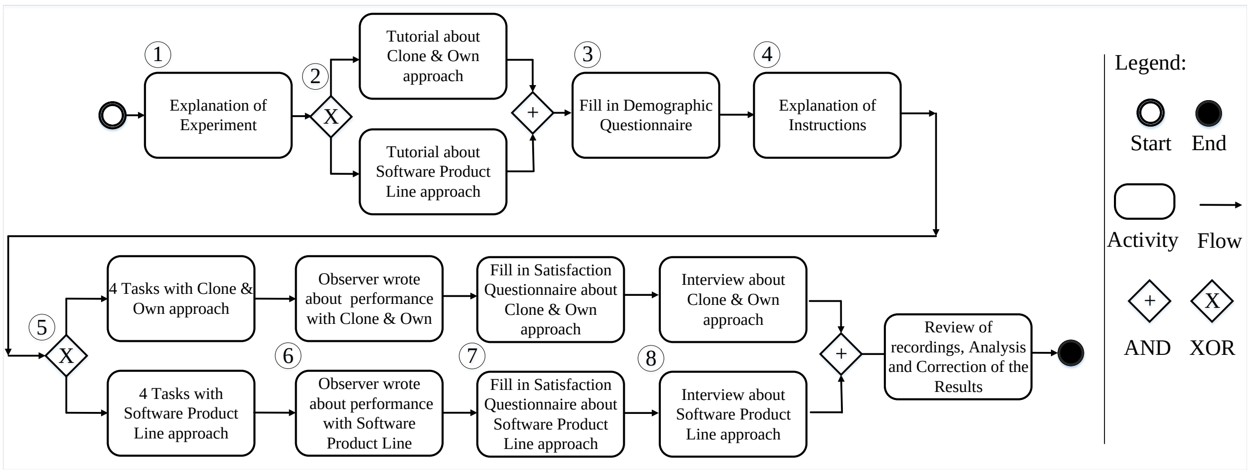
5. Results

According to our design, the most suitable statistical test is General Linear Model (GLM) [9]. The response variables for this test are Efficiency, Effectiveness, and Satisfaction; the Fixed Factor is the Method; the Random Factor is the subject since we need to represent the subject of each measure; the Covariable is the block variable Task since it is a variable that should not affect the response variable but that we must check. Conclusions extracted from GLM are supported with descriptive data through box-and-whisker plots and histograms.

The application of GLM depends on three assumptions: residuals are independent of each other; residuals must be normally distributed; residuals should have the same variance for all values of the factor (homoscedasticity assumption). We ensured that all of these assumptions were satisfied. All of the residuals obtained a value close to 2 using the Durbin-Watson tests, which means that residuals are uncorrelated. All of the residuals obtained a p-value higher than 0.05 with the K-S test, which means that residuals are normally distributed. All of the residuals obtained a p-value higher than 0.05 with Levene's test, which means that residuals have the same variances for each factor.

The effect size shows the magnitude of differences for each factor. It is usually applied when null hypotheses are rejected in order to study the level of significant differences between the treatment means. We calculated the effect size through Partial Eta Squared, which describes the proportion of the variability in the dependent measure that is attributable to a factor. The most common interpretation is: between 0.02 and 0.13 is a small effect; between 0.13 and 0.26 is a medium effect; more than 0.26 is a large effect [35].

The power of any statistical test is defined as the probability of rejecting a false null hypothesis. Statistical power is inversely related to beta or the probability of making a Type II error. In short, power = 1 - β. This value can be between 0 and 1. When the value is 1, it means that if the study is replicated 100 times, the probability of correctly rejecting the null hypothesis is 100%. According to G*Power [15], for an effect size of 0.9, α=0.05 and a power of 0.95, we need a sample size of 39 subjects. Since we are working with 10 subjects, we have the threat of a low statistical power. In order



1
Explanation of Experiment
2
X
Tutorial about Clone & Own approach
Tutorial about Software Product Line approach
+
3
Fill in Demographic Questionnaire
4
Explanation of Instructions
5
X
4 Tasks with Clone & Own approach
4 Tasks with Software Product Line approach
6
Observer wrote about performance with Clone & Own
Observer wrote about performance with Software Product Line
7
Fill in Satisfaction Questionnaire about Clone & Own approach
Fill in Satisfaction Questionnaire about Software Product Line approach
8
Interview about Clone & Own approach
Interview about Software Product Line approach
+
Review of recordings, Analysis and Correction of the Results
Legend:
Start
End
Activity
Flow
AND
XOR

Figure 4: Experimental Procedure

to minimize this threat, we measured the value of Effectiveness and Efficiency for each task independently. This way we increased our sample per treatment: 5 subjects*4 tasks results in 20 samples. Next, we analyze the power of each response variable independently.

5.1. Effectiveness

Figure 5(a) shows the results of the GLM test applied to Effectiveness. Since the p-value of Method is less than 0.05, we can conclude that there are significant differences between the two treatments (CaO and SPL). The effect size of 0.98 shows that the magnitude of this difference is large. The statistical power is the highest, which means that the rejection of the null hypothesis is not a false rejection. Since Task is a block variable, we are only interested in studying whether or not a specific task affects one of the methods. That is why we only analyzed the interaction Method*Task and we did not consider Task as a factor. For the interaction Method*Task, the p-value was higher than 0.05, so there were not significant differences in this case. This means that there was not a type of task used in the experiment that was affecting only a specific method.

Figure 5(b) shows the box-and-whisker plot for the response variable Effectiveness. It shows that the value for SPL is better than for CaO. The median, the first quartile, and the third quartile get better values for SPL. The two boxes are related by a line between two points that represent their averages. The average of Effectiveness for SPL is also better than for CaO. Figure 5(c) shows the histogram. Note that the frequency is higher than the number of subjects since we are considering each treatment (SPL and CaO) in four tasks. Therefore, we have a total number of 20 samples per treatment (5 subjects per treatment working with 4 tasks each). We conclude that an Effectiveness of 100% is more frequent in SPL than in CaO. We have 15 samples of 100% in Effectiveness using SPL and 10 samples using CaO. Moreover, Effectiveness lower than 60% only appears in CaO. The normal curve shows that values for Effectiveness tend to be high (around 80%).

According to our analysis, we can state that there are significant differences between SPL and CaO in terms of Effectiveness. The values for SPL are better than the values for CaO, obtaining values close to 100% Effectiveness. Therefore, we reject $H_{\mathbf{01}}$, which claims that the effectiveness working with SPL is the same as working with CaO.

5.2. Efficiency

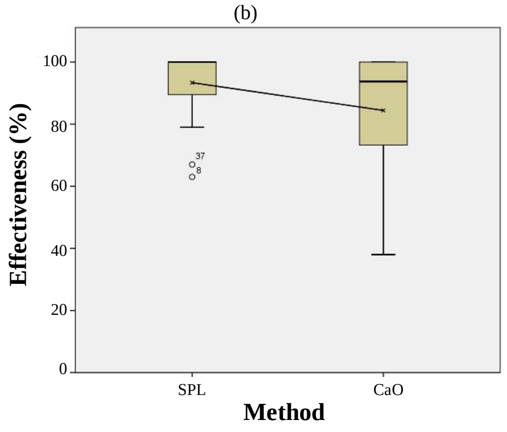
Figure 6(a) shows the result of the GLM test applied to Efficiency. The p-value is less than 0.05 for the factor Method, so there are significant differences between the two treatments. The effect-size of 0.94 is large, which means that this difference between the treatments is considerable. The statistical power is the highest, which means that we can reject the null hypothesis without error. The interaction Method*Task does not show significant differences since the p-value is higher than 0.05, so the type of task is not affecting the results for a specific method.

Figure 6(b) shows the box-and-whisker plot for the response variable Efficiency. The median, the first quartile, and the third quartile get better values for SPL. Moreover, the average for SPL (represented by the ends of the line that connect the two boxes) is clearly higher than for CaO. Figure 6(c) shows the histogram for Efficiency. It shows that values higher than 90 are exclusive

(a)

Variable	P-Value	Partial Eta Squared	Observed Power
Method	0.00	0.98	1
Method*Task	0.19	0.11	0.33

(b)



(c)

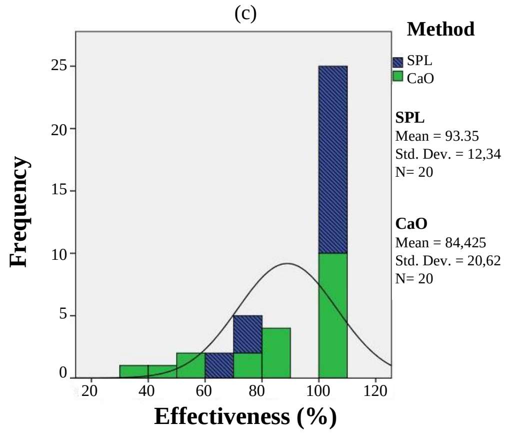
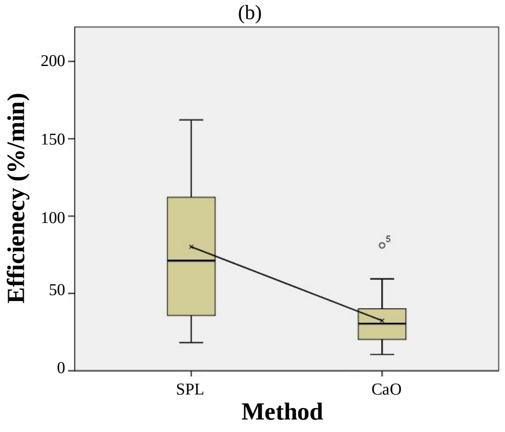


Figure 5: (a) Box-Plot for Effectiveness. (b) Histogram for Effectiveness. (c) Results after applying GLM to Effectiveness.

(a)

Variable	P-Value	Partial Eta Squared	Observed Power
Method	0.00	0.94	1
Method*Task	0.6	0.03	0.12

(b)



(c)

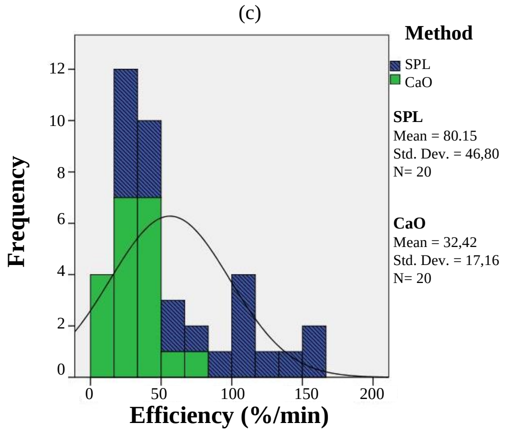


Figure 6: (a) Results after applying GLM to Efficiency. (b) Box-Plot for Efficiency. (c) Histogram for Efficiency.

to SPL. Most of the samples for CaO (18 samples) are between 10 and 40 for Efficiency, which is quite poor. The normal curve shows that values for Efficiency tend to be around 50.

According to our analysis, we can state that there are significant differences between SPL and CaO in terms of Efficiency. The values for SPL are better than the values for CaO, obtaining values higher than 90 exclusively for SPL. Therefore, we reject $\mathbf{H_{02}}$, which claims that the efficiency working with SPL is the same as working with CaO.

5.3. Satisfaction

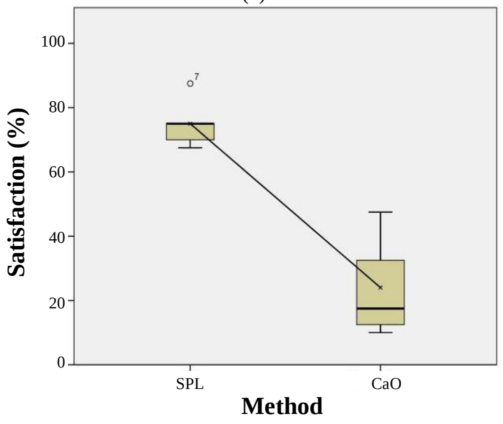
Figure 7(a) shows the result of the GLM test applied to Satisfaction. Note that satisfaction was measured after finishing the execution of all tasks, so we only have one measure of Satisfaction per subject. In this case, the block variable Task cannot be included in the statistical model since we did not distinguish the satisfaction for each task. The p-value in Figure 7(a) is less than 0.05 for the factor Method, which means that there are significant differences between the two treatments. The effect-size of 0.96 is a large effect, which means that these differences are considerable. The statistical power is the highest, which means that we can reject the null hypothesis without error.

Figure 7(b) shows the box-and-whisker plot for the response variable Satisfaction. The median, the first quartile, and the third quartile are clearly better for SPL. There is also a great difference between the averages of

(a)

Variable	P-Value	Partial Eta Squared	Observed Power
Method	0.00	0.96	1

(b)



(c)

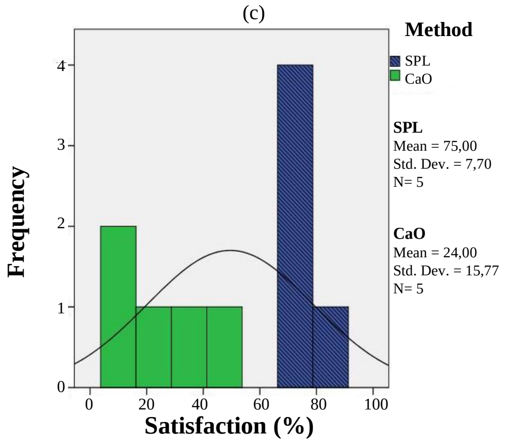


Figure 7: (a) Results after applying GLM to Satisfaction. (b) Box-Plot for Satisfaction. (c) Histogram for Satisfaction.

the two treatments (represented by the ends of the line that connects the two boxes). Figure 7(c) shows the histogram for Satisfaction. Note that since satisfaction was not measured per task but at the end of all of the tasks, we have reduced the number of samples per treatment to the number of subjects (5). There is no overlapping between the measures of Satisfaction for SPL and CaO. The 5 samples of SPL are above 70 and the 5 samples of CaO are between 10 and 50. The normal curve shows that measures are distributed equitably through all of the possible values between 0 and 100.

According to our analysis, we can state that there are significant differences between SPL and CaO in terms of Satisfaction. The values for SPL are better than the values for CaO. Therefore, we reject H_{03}, which claims that the satisfaction working with SPL is the same as working with CaO.

6. Discussion

The interviews by the instructor, the results presented in Section 5, and the recordings of the experiment allowed us to improve the knowledge of the results in order to compare the two approaches:

1. For both approaches (SPL and CaO), the subjects wanted to check the correctness of the task performed. Since this checking increased the time to finish the tasks, the efficiency decreased. These checking actions are used with higher frequency when new software products are created (tasks T1 and T2). The checking actions had higher frequency with the SPL approach. Specifically, in the SPL approach, the subjects checked both the correctness of the task and the SPL state; i.e., wrong features or mistakes in the software products. The time spent on the checking actions is higher in the SPL approach. However, efficiency is still better with this approach.
2. For the SPL approach, the subjects tried to control the impact on the SPL products with every modification. Specifically, they checked if a modification could negatively affect other software products. The subjects acknowledged that the SPL approach generated trust. For example, a subject who used the SPL approach declared *"The SPL gives me confidence, I doubt that it can develop any forbidden product configuration. To avoid this circumstance, I need to check the generated code"*. To improve the trust in their actions, three subjects (two with the SPL approach and one with the CaO approach) stated that they would like to control the changes performed in a software product, who has performed them, and the possibility of returning to a previous situation.
3. When the subjects stated their opinion about the CaO approach, they claimed that *"it is easy to use"*. In contrast, they declared *"There is too much information because the number of elements (induction hobs and their components) is very high"*. Similarly, two subjects commented *"I did not find the same difficulty in all of the processes; the difficulty increases when the amount of information I must manage is large"*. Thus, the subjects consider the CaO approach to be easy to use and efficient when the number of software products to manage is small. The subjects who used the CaO

approach considered *"a way to improve the management of the situations with a large amount of information could be to dispose of help about searching and traceability"*. The subjects agreed that current search capabilities are not up to the task to perform CaO in an industrial context.

4. The subjects want the approach used to develop their daily work performance in the best possible way. In this sense, one subject claimed *"I want a solution to generate real software products. I do not need a solution to generate very complicated configurations"*.
5. In both approaches, the subjects declared that they would like to be able to see if there are software products that have stopped being used. The subjects were not sure if the unused software should be eliminated or denoted in any way. In the case of the SPL approach, two subjects detected unused features as a consequence of the task performance.
6. It is remarkable the high value obtained from the variance for Efficiency with the SPL approach. The analysis of the subjects' performance shows that the subjects with minor values of Effectiveness spent more time to perform the tasks, producing an increase in the variance for Efficiency. In order to improve the adoption of the SPL approach, it will be necessary address in greater depth the decrease of Efficiency in concrete circumstances.

7. Threats to Validity

We use the classification of threats to validity in [45]. This classification distinguishes four aspects of validity:

Conclusion validity: This aspect is concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment.

- Low statistical power: To minimize this threat, we have used a confidence interval where conclusions are 95% representative. This means that if they followed a normal distribution, the results would be true 95% of the times.
- Error rate: To minimize this threat, we have reviewed the recordings to obtain the data. The variables depend on well-known metrics, and we have used the most suitable statistical test (GLM) according to our design to perform the statistical analysis.
- Reliability of measures: In order to minimize this threat, the measurements have been obtained from the recordings. Furthermore, this threat was alleviated by applying the same procedure to each individual experiment when the data was extracted and by using the same formula to calculate response variable values.
- Reliability of treatment implementation: To minimize this threat, the experiment was conducted in the same way with every subject.
- Random heterogeneity of subject: The experiment was affected by this threat because the knowledge and the background of the subjects were different to each other. The fact that all of the subjects were engineers with a high level of experience in industry reduced the threat considerably.

Internal validity: This aspect of validity concerns influences that can affect the factor with respect to causality without the researcher's knowledge.

- History: Different treatments were applied on different days; however, this threat was minimized because every subject only participated in the experiment on one day.
- Selection: Outcomes can be biased by the chosen subjects if the subjects are not suitable for the experiment. To minimize this threat, all of subjects were recruited by BSH staff, who have great experience in dealing with processes similar to the ones used in our experimental problem.
- Compensatory rivalry: This threat is minimized because the subjects did not know the outcomes achieved with the different treatments.
- Resentful demoralization: This threat occurs because only one approach is applied by each subject, so some subjects might prefer to work with the other approach. To minimize this threat, the subjects never know the outcomes of the other subjects who worked with the other approach.

Construct validity: This aspect of validity concerns generalizing the result of the experiment to the concept or theory behind the experiment.

- Mono-operation bias: The experiment is affected by this threat because we have used a different development tool for each approach (SPL and CaO). Note that the experiment is based on existing tools in the company and there is not a single tool that implements both approaches. The generalization of results to other tools must be made with caution.

- Mono-method bias: Experiments with a single type of measure can result in measurement bias [34]. Effectiveness and efficiency are affected by this threat. We minimize its effect using the recordings to calculate the time spent and the success of the task. This threat was avoided for satisfaction because the satisfaction questionnaire includes questions that are expressed in a both a positive and a negative way.

- Evaluation apprehension: When the subjects are evaluated, they may be afraid. To minimize this threat, at the beginning of the experiment, the instructor told every subject that it was not a test of their abilities.

- Hypothesis guessing: To minimize this threat, the subjects did not know the objective of the experiment.

- Task design: The proposed tasks do not have a true/false answer; it is very difficult for users to develop the software product correctly if they do not understand the task. On the other hand, the task statements are real tasks that were extracted from the daily work in the BSH induction hob division. This threat is minimized because the tasks are non-trivial. This threat was minimized because the tasks were non trivial. To correctly perform a task the software engineers should know the software products. The catalogue contains 46 software products with 81 features, each of the induction hob models is composed of more than 500 elements, including around 100 class elements on average. The values achieved for software engineers for the Effectiveness show that only one engineer with SPL approach and other engineer with CaO approach performed the tasks correctly, then the tasks are non-trivial.

External validity: This aspect of validity is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of relevance for other cases.

- Population: The number of subjects is not enough high to generalize the results to all companies. However, it is important to note that the role of the subjects (software engineers) makes an interesting contribution in an area where most experiments are conducted using students as subjects. Further work would be to replicate this experiment with a larger number of subjects in different environments in order to mitigate this threat.

- Interaction of selection and treatment: The role of subjects (software engineers) is interesting, but it would be necessary to replicate the experiment with different roles in order to mitigate this threat.

- Generalizability: Since this experiment has been conducted in a specific domain, we think that the generalizability of findings should be undertaken with caution. Other experiments in different domains should be performed to validate our findings.

8. Conclusion

The Cao and SPL approaches have been shown to be two good software development methodologies for software development in certain contexts. These contexts are when the software products to develop can be built from small variations in previous existing software products.

In the context of the software development for induction hobs of our industrial partner, BSH group, we have designed an experiment to compare software development using two approaches: SPL and CaO. We designed an experiment with a single factor (the approach used for software development). In this experiment where software engineers performed a series of tasks to develop software from legacy product software, we measured effectiveness, efficiency, and satisfaction.

Our results show that software engineers achieve better values for effectiveness, efficiency, and satisfaction with the SPL approach. Furthermore, the results show the following findings revealed by this work:

- The SPL approach is more efficient than the CaO approach. The SPL approach is more efficient even though this approach requires a larger number of checks than the CaO approach.

- The software engineers acknowledged that the SPL approach offers more possibilities those they need to perform daily tasks. Even though checking was required when they performed the tasks with the SPL approach, they trusted the SPL approach.

- The software engineers asked for better search capabilities from the CaO approach.

In the near future, we plan to replicate this experiment in other contexts and increase the number of subjects. These replications will allow us to generalize the results regardless of the domain.

Acknowledgments

This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the Project ALPS (RTI2018-096411-B-I00).

References

[1] ISO/IEC 25000:2014, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE). Standard ISO/IEC 25000:2014, International Organization for Standardization, Geneva, Switzerland, 2014.

[2] S. Adam and K. Schmid. *Effective Requirements Elicitation in Product Line Application Engineering – An Experiment*, pages 362–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[3] ANSI/NCITS. Ansi/ncits-354 common industry format (CIF) for usability test reports. Technical report, NIST Industry Usability Reporting, 2001.

[4] W. K. G. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering*, 22(6):2972–3016, Dec 2017.

[5] E. Bagheri and D. Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 19(3):579–612, 2011.

[6] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615 – 636, 2010.

[7] R. Bonifácio, P. Borba, C. Ferraz, and P. Accioly. Empirical assessment of two approaches for specifying software product line use case scenarios. *Software & Systems Modeling*, pages 1–27, 2015.

[8] J. Brooke. SUS: a retrospective. *Journal of usability studies*, 8(2):29–40, 2013.

[9] R. Christensen. *Plane Answers to Complex Questions The Theory of Linear Models*. Springer Science+Business Media, New York, USA, 2002.

[10] P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[11] K. Constantino, J. A. Pereira, J. Padilha, P. Vasconcelos, and E. Figueiredo. An empirical study of two software product line tools. In *ENASE 2016 - Proceedings of the 11th International Conference on Evaluation of Novel Approaches to Software Engineering, Rome, Italy 27-28 April, 2016.*, pages 164–171, 2016.

[12] D. Dermeval, T. Tenório, I. I. Bittencourt, A. Silva, S. Isotani, and M. Ribeiro. Ontology-based feature modeling. *Expert Syst. Appl.*, 42(11):4950–4964, July 2015.

[13] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki. An exploratory study of cloning in industrial software product lines. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 25–34, Washington, DC, USA, 2013. IEEE Computer Society.

[14] A. Durán, D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Flame: A formal framework for the automated analysis of software product lines validated by automated specification testing. *Softw. Syst. Model.*, 16(4):1049–1082, Oct. 2017.

[15] F. Faul, E. Erdfelder, A.-G. Lang, and A. Buchner. G* power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behavior research methods*, 39(2):175–191, 2007.

[16] S. S. M. Fauzi, M. hairul Nizam Nasir, N. Ramli, and S. Sahibuddin. *Software Process Improvement and Management: Approaches and Tools for Practical Development*. Information Science Reference, 2012.

[17] W. Fenske, J. Meinicke, S. Schulze, S. Schulze, and G. Saake. Variant-preserving refactorings for migrating cloned products to a product line. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 316–326, Feb 2017.

[18] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Castor Filho, and F. Dantas. Evolving software product lines with aspects: An empirical study on design stability. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 261–270, New York, NY, USA, 2008. ACM.

[19] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed. Enhancing clone-and-own with systematic reuse for developing software variants. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 391–400, Sep. 2014.

[20] J. Font, M. Ballarín, O. Haugen, and C. Cetina. Automating the variability formalization of a model family by means of common variability language. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, page 411–418, New York, NY, USA, 2015. Association for Computing Machinery.

[21] E. Ghabach, M. Blay-Fornarino, F. E. Khoury, and B. Baz. Clone-and-own software product derivation based on developer preferences and cost estimation. In *2018 12th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–6, May 2018.

[22] J. Gonzalez-Huerta, E. Insfran, S. Abrahão, and G. Scanniello. Validating a model-driven software architecture evaluation and improvement method: A family of experiments. *Information and Software Technology*, 57:405 – 429, 2015.

[23] V. Guana and D. Correal. Improving software product line configuration: A quality attribute-driven approach. *Information and Software Technology*, 55(3):541 – 562, 2013. Special Issue on Software Reuse and Product LinesSpecial Issue on Software Reuse and Product Lines.

[24] R. Heradio, H. Perez-Morago, D. Fernandez-Amoros, F. Javier Cabrerizo, and E. Herrera-Viedma. A bibliometric analysis of 20 years of research on software product lines. *Inf. Softw. Technol.*, 72(C):1–15, Apr. 2016.

[25] N. J. Juzgado and X. Ferré. How to integrate usability into the software development process. In *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, pages 1079–1080, 2006.

[26] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[27] D. Kieras. Using the keystroke-level model to estimate execution times. *University of Michigan*, 2001.

[28] H. Koziolek, T. Goldschmidt, T. de Gooijer, D. Domis, S. Sehestedt, T. Gamer, and M. Aleksy. Assessing software product line potential: an exploratory industrial case study. *Empirical Software Engineering*, 21(2):411–448, 2016.

[29] S. Krishnan, R. R. Lutz, and K. Goševa-Popstojanova. Empirical evaluation of reliability improvement in an evolving software product line. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 103–112, New York, NY, USA, 2011. ACM.

[30] J. Krüger, L. Nell, W. Fenske, G. Saake, and T. Leich. Finding lost features in cloned systems. In *Proceedings of the 21st*

International Systems and Software Product Line Conference - Volume B, SPLC '17, pages 65–72, New York, NY, USA, 2017. ACM.

[31] R. Lapeña, M. Ballarín, and C. Cetina. Towards clone-and-own support: locating relevant methods in legacy products. In *Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016*, pages 194–203, 2016.

[32] A. Marcolino, E. Oliveira, I. Gimenes, and T. U. Conte. Towards validating complexity-based metrics for software product line architectures. In *2013 VII Brazilian Symposium on Software Components, Architectures and Reuse*, pages 69–79, Sept 2013.

[33] B. Michalik, D. Weyns, N. Boucke, and A. Helleboogh. Supporting online updates of software product lines: A controlled experiment. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 187–196, Sept 2011.

[34] J. I. Panach, S. España, Ó. D. Tubío, O. Pastor, and N. J. Juzgado. In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction. *Information & Software Technology*, 62:164–186, 2015.

[35] C. A. Pierce, R. A. Block, and H. Aguinis. Cautionary note on reporting eta-squared values from multifactor anova designs. *Educational and psychological measurement*, 64(6):916–924, 2004.

[36] F. Pérez, M. Ballarín, R. Lapeña, and C. Cetina. Locating clone-and-own relationships in model-based industrial families of software products to encourage reuse. *IEEE Access*, 6:56815–56827, 2018.

[37] I. Reinhartz-Berger and A. Sturm. Comprehensibility of uml-based software product line specifications. *Empirical Software Engineering*, 19(3):678–713, 2014.

[38] J. Santos and U. Kulesza. Quantifying and assessing the merge of cloned web-based system: An exploratory study. In *The 28th International Conference on Software Engineering and Knowledge Engineering, SEKE 2016, Redwood City, San Francisco Bay, USA, July 1-3, 2016.*, pages 583–588, 2016.

[39] A. Schlie, D. Wille, S. Schulze, L. Cleophas, and I. Schaefer. Detecting variability in matlab/simulink models: An industry-inspired technique and its evaluation. In *SPLC*, 2017.

[40] L. P. Tizzei, M. O. Dias, C. M. F. Rubira, A. Garcia, and J. Lee. Components meet aspects: Assessing design stability of a software product line. *Information & Software Technology*, 53(2):121–136, 2011.

[41] T. S. Tullis and J. N. Stetson. A comparison of questionnaires for assessing website usability. In *Usability Professional Association Conference*, pages 1–12. Citeseer, 2004.

[42] G. Vale, D. Albuquerque, E. Figueiredo, and A. Garcia. Defining metric thresholds for software product lines: A comparative study. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 176–185, New York, NY, USA, 2015. ACM.

[43] K. Villela, A. Silva, T. Vale, and E. S. de Almeida. A survey on software variability management approaches. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 147–156, New York, NY, USA, 2014. ACM.

[44] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[45] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.