Evaluating the Influence of Scope on Feature Location*

África Domingo[a], Jorge Echeverría[a], Óscar Pastor[b], Carlos Cetina[a]

[a] *Universidad San Jorge. SVIT Research Group, Zaragoza, Spain*
[b] *Universitat Poliècnica de València. PROS Research Center, Valencia, Spain*

Abstract

Context: Feature Location (FL) is a widespread technique that is used to maintain and evolve a software product. FL is also helpful in reengineering a family of software products into a Software Product Line (SPL). Despite the popularity of FL, there is no study that evaluates the influence of scope (single product or product family) when engineers perform FL.

Objective: The goal of this paper is to compare the performance, productivity, and perceived difficulty of manual FL when scope changes from a single product to a product family.

Method: We conducted a crossover experiment to compare the performance, productivity, and perceived difficulty of manual FL when scope changes. The experimental objects are extracted from a real-world SPL that uses a Domain-Specific Language to generate the firmware of its products.

Results: Performance and productivity decrease significantly when engineers locate features in a product family regardless of their experience. For these variables the impact of the FL Scope is medium-large. On contrast, for perceived difficulty, the magnitude of the difference is moderate and is not significant.

Conclusions: While performance and productivity decrease significantly when engineers locate features in a product family, the difficulty they perceive does not predict the significant worsening of the results. Our work also identifies strengths and weaknesses in FL. This can help in developing better FL approaches and test cases for evaluation.

Keywords: Feature Location, Controlled Experiment, Model-Driven

*Partially supported by MINECO under the Project ALPS (RTI2018-096411-B-I00).

Email addresses: `adomingo@usj.es` (África Domingo), `jecheverria@usj.es` (Jorge Echeverría), `opastor@dsic.upv.es` (Óscar Pastor), `ccetina@usj.es` (Carlos Cetina)

1. Introduction

Feature Location (FL) can arguably be seen as one of the most frequent maintenance tasks undertaken by software engineers [1, 2, 3, 4, 5, 6] since software maintenance and evolution involves adding new features to programs, improving existing functionalities, and removing unwanted functionalities. To this end, it is essential that software engineers find the elements of software features.

FL can also help to reengineer a family of software products into a Software Product Line (SPL) [7, 8] since an SPL involves the formalization of features that are shared by the products. To do this, it is essential for SPL engineers to be able to find the elements of SPL features.

Both classic FL (cFL) and FL in SPL reengineering (rFL) target the elements of features, however, the scope is different, i.e., a single product versus a product family, respectively. One might think that rFL is a particular case of cFL where the first step is to select a product in the family that has the feature and then to perform cFL on that product. Thus, one might conclude that cFL and rFL have similar performance, productivity and difficulty. The engineers of our industrial partner actually believed this (see Section 2 Motivation of the experiment). However, to date, no experiment has been done to compare the scopes of cFL and rFL. This can help engineers to better understand the problems they encounter when locating features in a product family. Knowing these problems is relevant in order to be able to develop new automated or semi-automated FL approaches that could mitigate them.

In this work, we present an experiment that compares different FL scopes (single product and product families), when engineers locate features manually, in terms of performance, productivity, and perceived difficulty. The experimental objects are extracted from a real-world SPL that uses a Domain-Specific Language (DSL) to generate the firmware of its products. A total of 18 subjects (classified in two groups based on their DSL experience) performed the FL tasks of the experiment.

In the FL tasks, the subjects had to locate elements of different features. We use F-measure to measure performance as the percentage of a FL task solved by a subject. F-measure takes into account both the elements that a subject includes in its solution and are correct and those that it includes and are not correct. We use productivity as the percentage of an FL task solved by a subject per minute. The subjects rate the perceived difficulty of each task using a 7-point Likert scale, ranging from *too easy* to *too difficult*.

Our results shows how performance and productivity decrease significantly when engineers locate features in a product family regardless of their DSL experience. When the FL is in a product family, it is more difficult to locate the feature elements and the errors are propagated by the models throughout the family.

A more in-depth analysis shows how, for example, features with a larger size get better performance. Dispersion is the feature characteristic that best explains the changes in performance that occur when the scope changes: the stronger the dispersion, the lower the performance and the greater the differences in performance when locating features in a single product or in a product family.

Our results also indicate a paradox: even though performance and productivity are significant worst when the subjects locates features in a product family, the difficulty they perceive does not predict the significant worsening of the results. The initial inclination of subjects is to think that locating features in a product family only takes more time than locating features in a product. However, it turns out that, in the context of a product family, new challenges arise with regard to feature propagation, the elimination process, and feature dispersion. Apparently, the subjects are not aware of the tricky challenge that FL presents in SPL reengineering.

The paradox introduced in the experiment is important because it could help engineers to understand and better perform reengineering Feature Location (rFL). If engineers realized the real difficulty of rFL, they might leverage this information to apply and develop techniques that would improve the performance and productivity of rFL tasks. Engineers perform classical Feature Location (cFL) tasks frequently (i.e., to remove unwanted functionalities in a feature). When they decide to reengineer a family of software products into an SPL, they need to perform rFL, and, by inertia, they may think that cFL is similar to rFL (i.e., they just have to choose a product, perform cFL, and check other products). However, there might be thousands of products in industrial settings and an engineer cannot comprehensively check all of them. The products they choose to locate the feature in can influence the results. This makes rFL more of a challenge than cFL without the engineers realizing it.

We hope that our work will raise awareness of the challenge posed by FL in SPL reengineering and that what we have learned will help design new approaches that compensate for the weaknesses of the engineers.

The rest of the paper is organized as follows: Section 2 describes the context that motivates the experiment. Section 3 reviews the related work, and Section 4 provides the necessary background in FL in product models and the software product domain. Section 5 describes the design of our experiment. Section 6 reports the results that we discuss in Section 7. Section 8 describes the threats to validity. Finally, Section 9 concludes the paper.

2. Motivation of the experiment

For more than ten years, the engineers of our industrial partner, BSH, have developed software that controls its induction hobs. BSH is one of the largest manufacturers of home appliances in Europe. Its induction division produces induction hobs that are sold under the Bosch and Siemens brands, among others. During those years, BSH has developed a family of software products and regularly carried out tasks of classic Feature Location (cFL) to modify the functionalities of its products. In cFL, an engineer has a description of the feature to be located and the product in the product family that contains this feature. For example, in such a product, there may be an unwanted functionality that must be located in order to eliminate it and replace it with the correct functionality.

Figure 1 illustrates a concrete example of cFL in BSH. To simplify, an induction hob (product) is composed of several hob plates (the place where the pots and pans get hot). These require: inverters, which generate the energy for the hob (triangles); inductors, which convert the energy into heat (circles); power channels, which transfer the energy from one element to other (lines); and power managers, which control the path followed by energy through the channels (rectangles).

In a concrete product (P1 in Figure 1), the users had reported that when cooking with a pot on the upper hob plate with medium power, the hob turns itself off after a while. To fix this, the engineers must first locate the relevant elements for this unwanted functionality. The engineers have the description of the unwanted functionality and the product where it is located as the input. To perform cFL, the engineers inspect the properties of the elements that they consider relevant to the description of the feature. Using the information in the description, the engineers identify the inverters and channels on the upper hob plate as being relevant to that unwanted functionality (a_1, a_2, c_1, and c_2

in Figure 1). After the cFL task, the engineers make the decision to replace these elements with a single inverter and a higher power channel (Figure 1, right).

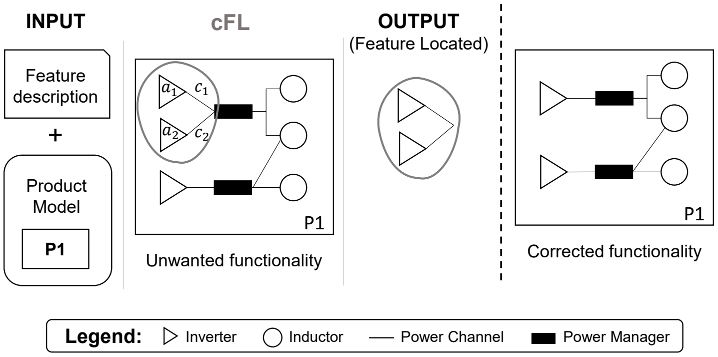

Figure 1: Example of classical Feature Location

At some point, the engineers of BSH decided that they want to start a Software Product Line (SPL) from the products that they already have. Software product lines are attractive because the goal is to reduce the development cost and time to market while improving the quality of the software systems by exploiting commonalities and managing the variability across a product family [9]. In an SPL, the common and variable features among the products are formalized to be reused in a systematic way in future developments.

To reengineer a family of software products into an SPL, engineers must identify the commonalities of their products and formalize them as reusable assets. Engineers must perform SPL reengineering Feature Location (rFL) tasks to formalize the features that capture the variability of the systems. In rFl, an engineer has the text description of the feature to be located and a set of products where the feature could be found. For example, in several products of a software family, there is a functionality that must be located in order to formalize it as a feature of the software product line.

When the engineers of our industrial partner (who have performed cFL regularly for years) have to perform rFL, their procedure is as follows: 1) they select a product in the family where they think the feature is located; 2) they perform cFL on that product; and 3) they search for that feature in the other products. The inertia of having performed cFL for so many years has influenced the way the engineers of our industrial partner perform rFL. For this reason, they argue that performing cFL and rFL have similar performance, productivity, and difficulty. After all, for them, the rFL task is essentially a cFL task where you must choose the product on which to perform cFL.

Figure 2 illustrates a concrete example in BSH where the engineers perform rFL in order to locate the feature of Double Hot Plate. The description of the Double Hot Plate feature is as follows: some induction hobs have a plate with an extra cooking space (called Double Hot Plate feature) that allows users to select a small or a large hob plate based on their needs. When a user selects the larger plate, two inductors are used, and more energy is needed to heat the plate without reducing the temperature that the plate must reach or the time in which it must heat up.

To locate the Double Hot Plate feature, the first step is to select a product model from the family of products. In industrial environments such as BSH that have product ranges that are differentiated by brands and are adapted to different countries, the family has more than a thousand products. This makes a thorough review of all of the products a time-consuming or even impossible task. The output of the rFL may be different depending on the products the engineers select to locate the feature. The following are examples of this:

> **Example A:** The engineers select P2 to locate the feature (the fist step of Example A of Figure 2). When they locate the feature (the second step), they might think that the feature is composed of a power manager (d_1 in Example A), two power channels (c_1 and c_2 in Example A), and two inductors (b_1 and b_2 in Example A). In the third step, the engineers search for that feature in other products, and if they select a product that confirms their hypothesis (e.g., product P3 in Figure 2), then they conclude that the feature is composed of the five elements they had found in the second step.
>
> **Example B:** The engineers select P1 in the first step (Example B of Figure 2). When locating the feature in the second step, they might think that the feature is composed of a power manager (d_1 in Example B), three power channels (c_1, c_2, and c_3 in Example B), and two inductors (b_1 and b_2 in Example B). In the third step, if they select a product that confirms their hypothesis (P4 in Example B), they conclude that the feature has six elements as they had hypothesized in the second step.

Example C: The engineers select P1 in the first step. They might think that the feature is composed of six elements as in Example B. In this example, the engineers select other products to confirm their hypothesis (P2 and P3 in Example C, step 3), which could make them change their first selection of elements to conclude that the feature is composed of five elements as in Example A.

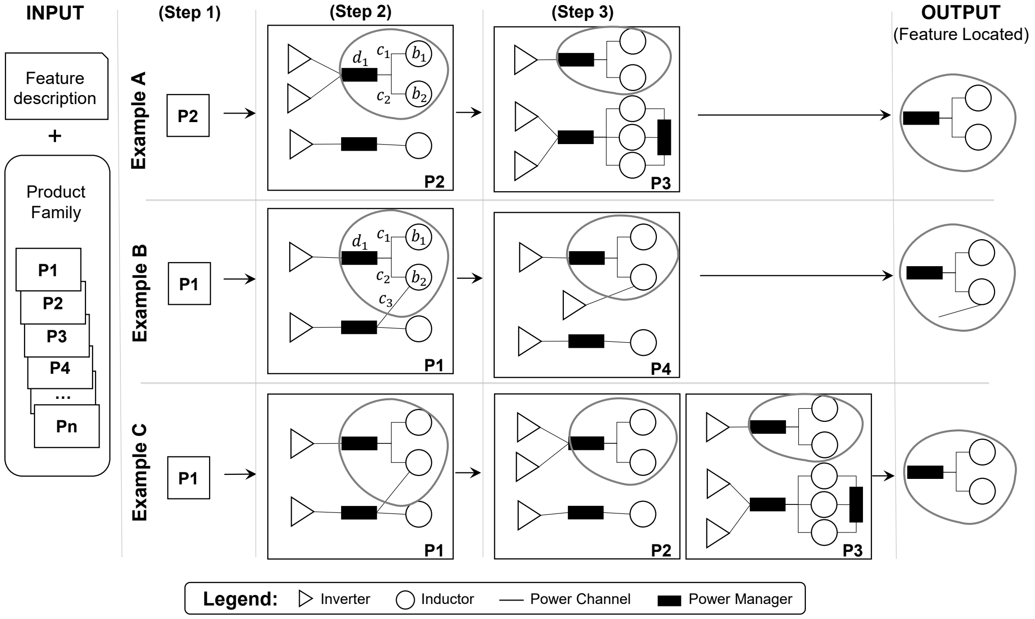

Figure 2: Example of reengineering Feature Location

Actually, it is true that rFL and cFL may look similar. In fact, once the product is known, the core of the localization task is the same: the engineers inspect the properties of the model elements to identify those model elements that are the most relevant to the feature description. That is why BSH engineers think that the difficulty of rFL and cFL is the same. However, the selection of products for both starting to locate a feature and for checking the results of the first search affects the performance and the productivity of the rFL tasks with respect to the cFL tasks without the engineers being aware of this. As Table 9 shows, in rFL, the engineers can fail in selecting the product, and they can propagate errors in the location of features. Also, the elimination process works worse in rFL than in cFL, and the size and dispersion of the features influence the result of rFL.

This SPL initialization is not only limited to the industrial partner's engineers. According to the results of a survey by Berger et al. [10], starting an SPL from existing products is widespread in industry. Those engineers who have previously maintained and evolved a family of products (and thus performed cFL) will have to perform rFl to initialize an SPL from this family of products. Thus, they have also been influenced by their experience of performing cFL before rFL. Consequently, they may also think that cFL and rFL have similar performance, productivity, and difficulty.

Our experiment has revealed that the subjects, BSH engineers (experts) and developers without domain-specific knowledge (non-experts), underestimate the rFL task. On the one hand, the subjects state that they do not perceive differences in difficulty between cFL and rFL, which aligns with what the engineers of our industrial partner stated before the experiment. On the other hand, the results for performance and productivity are significantly worse in rFL than in cFL, contrary to what BSH engineers believed. This misperception regarding the rFL task makes the already difficult initialization of an SPL in an industrial environment even more complicated

3. Related work

Researchers have conducted several studies to improve the understanding of features and to develop effective techniques to locate features. Some focus on automated techniques to support developers when locating features. Others focus on how developers locate features. The first are referred to as automated FL and the second ones are referred to as manual FL. We also find the term semi-automated FL because features are being located by developers that use different kinds of tools to automatize the process. Not only it is necessary to develop effective FL automated techniques, but it is also important to improve the way of using these techniques by the developers.

A survey of manual FL by Krüger et al. [25] reviews the literature about developers performing manual feature location. They compiled case studies and field studies from 2003 to 2018 that show how manual feature location is performed in industry. Krüger et al. argue that regardless of whether automated or semi-automated techniques are used, manual feature location is relevant. Most of these techniques require a seed to be able to locate the feature and this seed must be located manually. The seven works analyzed in the survey,[11, 12, 13, 3, 14, 15, 1] use a single software product for the

Table 1: Empirical studies on Feature Location

Work **Year**	**Focus**	**FL Scope**	**Systems**	**Variables**	**Strategy** **Context**	**Sample size**
Wilde et al. [11] 2003	Manual	Single Product	FORTRAN	Effort to adopt FL Techniques	Case study Industry	4 subjects with different experience
Revelle et al. [12] 2005	Manual	Single Product	C and Java	Concern overlap and Spread	Case study Academia	2 researchers
Wang et al. [13] 2011	Manual	Single Product	Java	Frequencies of physical actions	Experiment Academia	2 developers 18 students
Wang et al. [3] 2013	Manual	Single Product	Java	Frequencies of physical actions Precision, Recall, and F-measure	Experiment Academia	2 developers 54 students
Jordan et al. [14] 2015	Manual	Single Product	COBOL DSLs	Search relevance and success	Case study Industry	2 software engineers
Damevski et al. [15] 2016	Manual	Single Product	Visual Studio	Number of queries, terms in queries, and navigation command events	Field study Industry	667 developers
Krüger et al. [1] 2018	Manual	Single Product	C	LOC, Scattering Degree, and Tangling Degree	Case study Academia	2 researchers
Martinez et al. [8] 2015	Automated	Product Family	UML	Precision, Recall, and Variability safety	Case study Academia	No subjects
Assunção et al. [7] 2020	Automated	Product Family	UML	Number of common model elements	Case study Academia	No subjects
Font et al. [16] 2016	Automated	Product Family	DSL IHs	Precision, Recall, and F-measure	Case study Industry	No subjects
Marcén et al. [17] 2017	Automated	Single Product	DSL	Precision, Recall, and F-measure	Case study Industry	No subjects
Pérez et al. [18] 2018	Automated	Product Family	DSL	Precision, Recall, and F-measure	Case study Industry	No subjects
Font et al. [19] 2017	Automated	Product Family	DSL IHs	Matthew's Correlation Coefficient (MCC) Precision, Recall, and F-measure	Case study Industry	No subjects
Arcega et al. [20] 2015	Automated	Product Family	DSL PervML	Accuracy	Case study Academia	Masters' student as software engineer
Cetina et al. [21] 2017	Automated	Product Family	DSL	Recall, Precision, Area Under the Receiver Operating Characteristics curve, and MCC	Case study Academia	No subjects
Ballarín et al. [22] 2018	Automated	Product Family	DSL	Precision, Recall, F-measure, and MCC (Size, volume, density, multiplicity, and dispersion to report the location problem)	Case study Industry	No subjects
Pérez et al. [23] 2019	Semi-automated	Product Family	DSL	Precision, Recall, and F-measure	Case study Industry	19 Domain Experts
Pérez et al. [24] 2020	Manual, Automated	Single Product	DSL	Performance, Productivity, and Satisfaction	Experiment Industry	18 subjects
This work 2020	Manual	Single Product Product Family	DSL	Performance (F-measure), Productivity, and Perceived difficulty	Experiment Industry	18 subjects 13 developers 5 Domain Experts

feature location tasks and the software products are code developed. Table 1 includes some characteristics of these works.

The case study of Wilde et al.[11] analyzes the advantages and disadvantages of three techniques for locating features in C code when using them to locate features in FORTRAN Code. Two of the techniques are automated, and the third one is considered manual (grep text search method). An experienced academic programmer and a graduate student use software reconnaissance to locate a set of features. The experienced programmer, with knowledge about the software system used dependency graph method and the experienced FORTRAN programmer, used the grep text search method to locate the set of features.

The paper of Revelle, Broadbent, and Coppit [12] analyze the identification of the code associated to concerns. A concern is a notion that is more flexible than the notion of feature, but it does includes features. They propose guidelines to identify concerns and locate the code associated to them. They compare the percentage of overlap between the sets of code that each subject associates to the same concern. They also adapt Lai and Murphy's spread metric [26] to take into account the number of different files that contain code from the same concern. They conclude that the percentage of overlapping between the sets of code associated to a concern given by different researchers decreases when the number of files containing the code increases, when the spread increases.

The two papers of Wang et al. [13, 3] collect the results of three FL controlled experiments. Two full-time developers and 54 students participate in their experiments. In their first experiment they establish a heuristic to describe phases, patterns, and actions in a feature location task that they present in their first paper. In this second and third experiments, they refine their heuristic and analyze factors such as specific knowledge about feature location, task properties, or experience, which may influence the performance of developers when locating features. Their statistical analysis shows how specific knowledge about feature location phases, patterns and actions improve the developers' performance locating features.

The case study of Jordan et al. [14] has an industrial context. They analyze the search actions and the tools used by nomads, who are experienced software engineers that work on large software systems. They measure the percentage of relevant search results that are viewed by the subjects and use a search success indicator for the searches. A non-empty set of results, with manageable size, and containing relevant results is considered a successful search. They affirm that nomads are twice as effective locating features as the non-experts reported in other studies [6, 27].

Damevsky et al. [15] report a field study of how developers perform feature location during their daily activity. They report what type of queries, retrieval strategies, and patterns are used by developers when locating features. They also report the search tools used by developers to locate features and how often developers use them. They use the heuristic defined by Wang et al.[3] and analyze the tracking data of developers when locating features.

The case study on the Marlin 3D Printer of Kruger et al.[1] is focused on identifying and locating optional and mandatory features. They explore and compare the characteristics of these features, describing them through metrics such as Lines of Cod (LOC), Scattering Degree (the number of locations where the feature is implemented), or Tangling Degree (the number of features contained in a feature). The authors also provide a set of feature fact sheets that could be used in empirical studies about feature location.

The survey elaborated by Julia Rubin and Marsha Chechink [28] analyzes 24 automated FL techniques and explores how to adapt them to feature location in family products. They affirm that none of the existing FL techniques consider families of related products explicitly. To adapt the FL techniques to a product family, each one of the products of the family is considered an independent entity. They propose that by considering the relations of the products in a family, product line commonalities and variations can provide additional input to improve the accuracy of FL techniques.

Other works ([8, 7]) propose automated approaches for the feature identification and subsequent generation of SPLs from a set of existing software variants. Martínez et al. [8] use two real-word systems, with seven and three model variants, respectively, to evaluate their approach. They compare the number of model elements of the primitive system with the elements of the blocks identified in the model-based SPL generated. They affirm that the generated SPL can regenerate the previous variants and generate new valid variants. Assunção et al.[7] presents an automated approach to aid the generation of SPLs from existing UML class diagrams and the list of associated features. They generate a Feature Model, which expresses common and variable characteristics of a product family, and a Product Line Architecture, which allows developers to generate, maintain, and evolve system families. To evaluate their approach, they analyze the results obtained with 10 different applications. In each one of these applications, they use a set of well-defined features and apply a Genetic Algorithm to locate features in the Feature Model generated. Optimal values of precision, recall, and variability safety are obtained when the search results on the Feature Model and the input features are compared. Both studies use UML models as input instead of files of code, but neither of the two studies considers real developers using their approaches.

Feature location in models at the industrial scale is a central topic in previous works from our SVIT research group [16, 17, 18, 19, 20, 21, 22, 23, 24]. Given a feature description as input, these works [16, 17, 18] rank the model fragments that are relevant for the feature and explore different approaches to guide the automated feature localization: clustering (through Formal Concept Analysis) [16], empirical learning (through Learning to Rank) [17], and combinations of Similitude, Understandability, and Timing (through Latent Semantic Indexing, Model Size, and Defect Principle, respectively) [18]. In [19], the fitness function is fixed (Similitude), and five search strategies are evaluated (the Evolutionary Algorithm, Random Search, steepest Hill Climbing, Iterated Local Search with restarts, and a hybrid between the Evolutionary algorithm and Hill Climbing). In [20], models at run-time are proposed to be used for increasing the information for feature location. In [21], the sustainability of long-living software systems is exploited to guide feature location. The study of Ballarín et al. [22] provides measures to describe model fragments such as the ones for feature location in models. Collaborative FL is introduced in [23] when the FL task is complex and significantly exceeds the knowledge of a single software engineer. In [24], manual FL and automated FL are compared. The work shows that the subjects are equally satisfied with the results of the two approaches. Hence, we face the possibility that, if subjects consider the results to be good enough, they may lack the motivation to iterate on the query or the results through the usage of guided techniques.

The works about manual FL use one single software product to locate features in code. Most of the studies about automated FL also focus their attention on locating features in single software products (mainly in code), and the studies about feature location that take into account system variants or product model families do not take into account the developer who locates the features.

Our work addresses the research gap in manual FL in model products, comparing *Performance, Productivity, and Perceived difficulty* between locating features in a Single Product Model and locating features in a Product Model Family.

4. Background

4.1. Feature Location

Let us consider a software product that is composed of elements from a specific universe:

$$U = \{e_1, e_2, .., e_i, ..., e_j, ..., e_k, ..., e_n\}$$

These elements could be different depending on which artifacts we analyze (e.g., code, requirements or models). If we analyze code lines, the elements could be each word, or each line, or specific groupings of lines, depending on the granularity of the problem. If we analyze requirements or models, these elements also change.

Assuming the artifact and the granularity are fixed, we can now consider a software feature that is represented by a subset of elements of the universe that implements some functionality, for example:

$$F = \{e_i, e_j, e_k\}$$

Thus, we can understand a software product to be a set of elements of U:

$$P = \{F_1 = \{e_1, e_2\}, ..., F = \{e_i, e_j, e_k\}, ...F_N = \{e_i, ..., e_j\}\}$$

In this case, it is not very difficult to find the feature in the software product. The problem begins when the following occur: a software product changes, the features that compose a software product evolve, and the information about them is not as structured. Sometimes the software product is more similar to the one below:

$$P = \{e_1, \{e_2, e_i\}, ..., e_j, e_1, ..., e_k, e_i, e_j, e_k, ..., e_m, ..., e_r\}$$

We do not always have all of the information about the feature and we look for sets like $F = \{e_i, x, e_k, y\}$, where x and y are elements from the specific universe, but they are unknown elements in the feature location. This set is determined from a feature description, which may be more or less precise. Feature Location is the process of finding subsets of elements (not all of which are known), in a collection of elements that represents a software product, given some artifacts and a determined granularity of the problem. Artifacts and granularity specify the universe, U, to which the elements belong. The way the different elements of the universe are composed to build a software product, P, or features, F, depends on them and also on the domain of the problem.

FL can be classified as manual, semi-automated or automated depending on the degree of intervention required by a person to determine and to locate the set of elements that represents a feature. This experiment focuses on manual FL.

4.2. Feature Location in Product Models

When models are used to develop software, as in Model Driven Development (MDD) [29], the main development artifact is the product model. In MDD contexts, engineers specify the system to be built with the models and then obtain the source code through model transformations. When models are the main development artifact, FL is performed on product models. This is the case of our industrial partner, which uses a DSL to specify the firmware of its induction hobs. The C++ code that controls its induction hobs is obtained with a transformation from model to code. There are also MDD contexts in which the models are interpreted. An example of interpreted models is the case of Kromaia, a commercial video game where all of its elements are specified with a DSL that is interpreted at run-time [30]. In these cases, no code is generated from the models, so FL is performed on a product model.

Since a product model is a kind of software product, it can be represented by a collection of elements from a universe set as we represented a software product above. The model elements are the elements of a universe, and a feature is a set of model elements. Locating a feature in a product model means determining the set of model elements that compose the feature and finding these elements in the set of product model elements.

A Product Model Family is composed of several Single Product Models that are similar but different in order to meet certain market needs, i.e., a family of collections composed of model elements from the same universe. The set of elements that represent a feature can be in a single model or in several models. To locate a feature in a Product Model Family, it is necessary to check the product models in the family.

Induction Hobs constitute a Product Model Family for our partner. There are thousands of product models in this family, depending on the brand and the induction hob characteristics (i.e., the number or size of the hob plates, type of hob plates, etc). All of these product models are composed of elements from the same DSL that represent inverters, inductors, power channels, and power managers. The set of elements that represents a feature (i.e., a hob plate with extra coocking space) is in several models but not in all of them.

The Induction Hob universe of elements in BSH would be something like this:

$$U = \{a_1, a_2, .., a_{n_a}, b_1..., b_{n_b}, c_1..., c_{n_c}, d_1..., d_{n_d}\}$$

where a_{i_a}, b_{i_b}, c_{i_c}, and d_{i_d} represent specific inverters, inductors, power channels, and power managers, respectively: n_a being the number of inverters with $1 \leq i_a \leq n_a$; n_b being the number of inductors with $1 \leq i_b \leq n_b$; n_c being the number of power channels with $1 \leq i_c \leq n_c$; and n_d being the number of all of the inverters with $1 \leq i_d \leq n_d$.

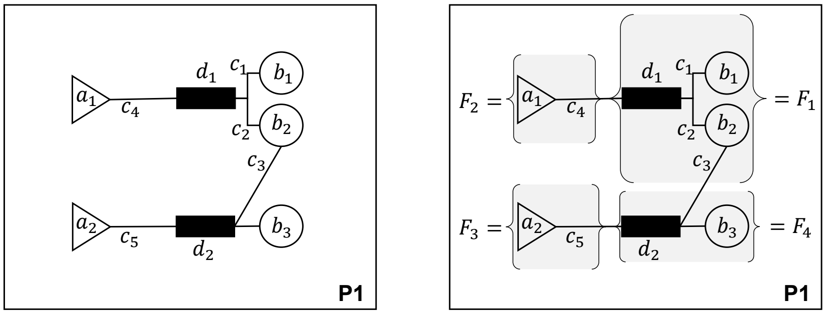

Figure 3: Abstract representation of a Product Model in Induction Hobs Family

Figure 3 shows the abstract representation of the product model P1 used in Section 2 (See Figures 1 and 2). The product model P1 could be represented as a a subset of elements of the universe U as follows:

$$\text{P1} = \{a_1, a_2, b_1, b_2, b_3, c_1, c_2, c_3, c_4, c_5, c_6, d_1, d_2\}$$

The Double Hot Plate feature of the rFL example (a hob plate with extra cooking space) could also be represented as a subset of the universe U:

$$F_1 = \{b_1, b_2, c_1, c_2, c_3, d_1\}$$

Therefore, the product model could be represented as a set of features as follows:

$$\text{P1}=\{F_1=\{b_1, b_2, c_1, c_2, c_3, d_1\}, F_2=\{a_1, c_4\}, F_3=\{a_2, c_5\}, F_4=\{b_3, c_6, d_2\}\}$$

where F_2 and F_3 represent different kinds of Simple Inverter features and F_4 represents a Simple Hob Plate feature.

4.3. Software Product Model Domain

In this experiment, we have used a subset of the Induction Hob Product Model Family from our industrial partner, BSH, which uses a Software Product Line (SPL) and a Domain-Specific Language (DSL) to generate the firmware of its induction hobs. This DSL is composed of 46 meta-classes, 74 references among them, and more than 180 properties. In order to gain legibility and due to intellectual property rights concerns, a simplified subset of the DSL is used in this experiment. We considered four kinds of model elements. Each one of them is characterized by three or four properties with different ranges of values depending on their functionality and the requirements of the induction hob. The metamodel and the graphical syntax of the DSL elements are shown in Figure 4.

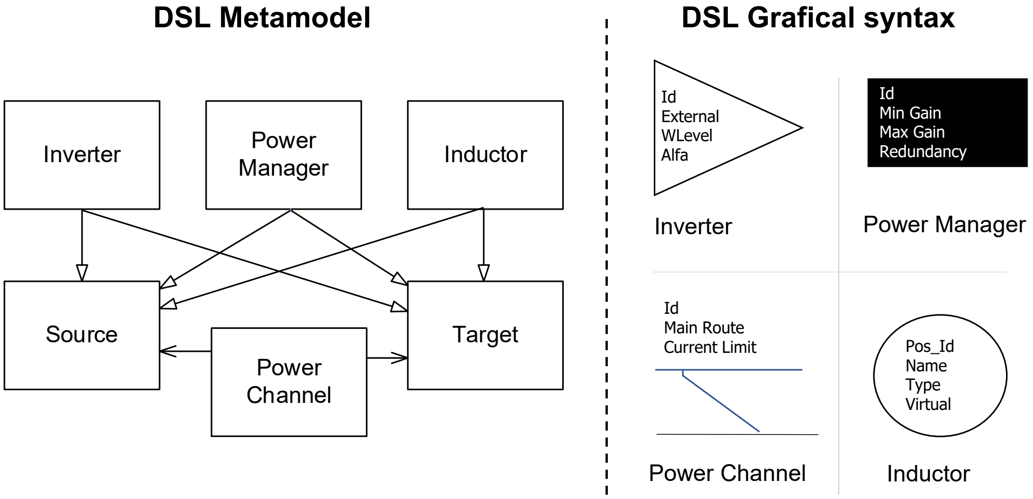

Figure 4: Metamodel and Graphical syntax of the DSL

The triangle represents an inverter. The inverter is in charge of converting input electric supply to the specific requirements of the induction hob. The power channels are represented as lines and they are in charge of transferring the energy from the inverter to the inductor. The power manager is represented by a rectangle and controls the path followed by the energy through the channels. The inductor is represented by a circle and is in charge of transforming energy into an electromagnetic field. These elements allow different characteristics included in an induction hob to be expressed. Each induction hob is represented by a set of elements, a product model. The elements and subsets of elements also allow different features included in an induction hob to be expressed.

Similarly to other software artifacts, the granularity can vary depending on the nature of the models and the features being located. Taking into account the MOF standard from the Object Management Group (OMG) to define a universal meta-model for describing modeling languages [31], we divide the relevant elements of a model into a set of atomic elements (*Class*, *Reference*, and *Property*), and we do not consider further subdivisions of those units in this work. *Class* is the core element. It holds a set of properties and associations (e.g., see the Inductor class element in the metamodel in Figure 4 (top-left)). *Reference* relates two class elements. It includes a source class and a target class, a multiplicity for the target class and the source class, and a name. Associations can also be distinguished by whether or not they are containment associations. For instance, the Inductor class reference in the metamodel of Figure 4 (top-left) is a containment reference whose source is the Induction Hob class (multiplicity 1) and whose target is the Inductor class (multiplicity any), while the source reference is a reference (non-containment) whose source is the Provider Channel class (multiplicity 1) and whose target is the Inverter class (multiplicity 1). *Property* gives information about a class. It includes the property meta-name, the type, and the value. For instance, the Inverter class element in the metamodel in Figure 4 (top-left) contains a property named WLevel whose type is a String. Based on this division, a feature is a subset of the model elements that are present in the model, with the granularity of the elements being classes, references, or properties.

The features and the product models used in this experiment are collections of model elements with specific properties. Figure 5 shows the graphical representation of the realization of the feature (F) described by the text 'High power external supply for induction chain' in a Product Model (PM) from the Product Model Family used in the experiment. Ordered numbers have been assigned to each element in the product models to easily identify each model element during the experiment.

5. Experiment design

5.1. Objectives

We have organized our research objectives using the Goal Question Metric template for goal definition following the the guidelines for reporting software

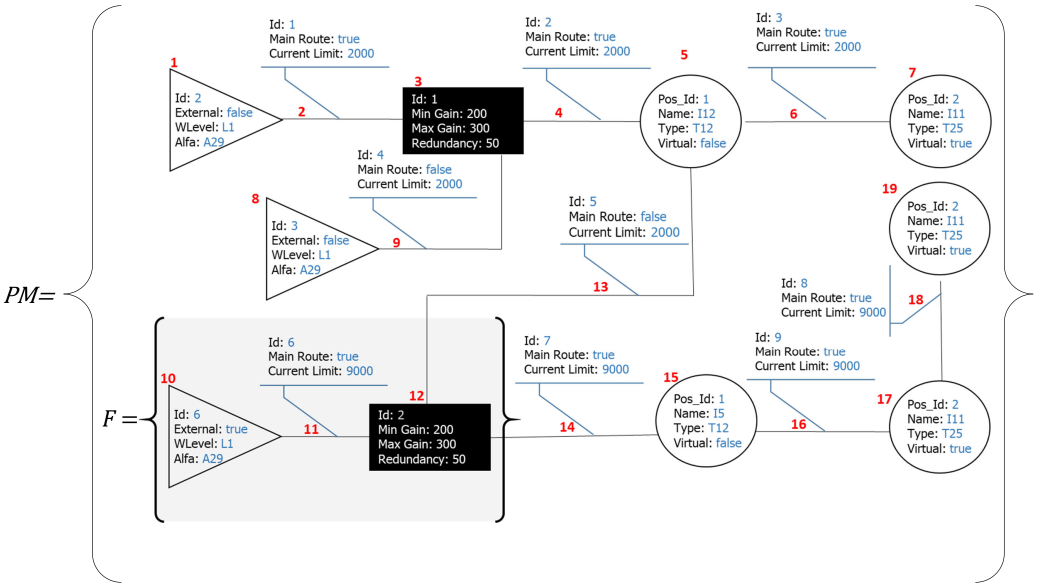

Figure 5: Graphical representation of a Product Model and a Feature in the DSL

engineering experiments in [32]. Our goal is to:

Analyze Different *FL Scopes* (Single Product Models and Product Model Families) **for the purpose of** comparison, **with respect to** *Performance*, *Productivity*, and *Perceived difficulty*, **from the point of view of** experts and non-experts in the specific domain, **in the context of** manual FL in models in a company.

5.2. Research questions and hypotheses

Through this experiment, we seek to answer the following three research questions about manual FL in models:

RQ1 Does the *FL Scope* used for locating features impact *Performance* in manual FL? The corresponding null hypothesis is $H_0 1$: The *FL Scope* does not have an effect on *Performance*.

RQ2 Does the *FL Scope* used for locating features impact *Productivity* in manual FL? The null hypothesis for *Productivity* is $H_0 2$: The *FL Scope* does not have an effect on *Productivity*.

RQ3 Is the *Perceived Difficulty* different when subjects use different *FL Scopes* for locating features? To answer this question we formulated the null hypothesis $H_0 3$: The *FL Scope* does not have an effect on *Perceived Difficulty*.

The hypotheses are formulated as two-tailed hypotheses since we do not know of empirical or theoretical studies that support a specific direction for the effect of scope.

5.3. Experimental objects

In this experiment we have used a Product Model Family made up of five product models, that are characterized by the number of model elements and the number of total properties in each one. Table 2 shows this information. These Single Product Models are the collections of model elements where the subjects had to locate features during the experiment. Figure 5 shows the graphical representation of Product Model 4 (PM4) from the Product Model Family used in the experiment.

Table 2: Product Model Description

	Nº elements	**Nº properties**
Product Model 1 (PM_1)	18	64
Product Model 2 (PM_2)	14	49
Product Model 3 (PM_3)	15	53
Product Model 4 (PM_4)	19	67
Product Model 5 (PM_5)	15	53
Product Model Family (PMF)	81	286

The subjects who participated in the experiment had to locate features in a Single Product Model (SPM) and in a Product Model Family (PMF), which represent two different *Feature Location Scopes*. When the subjects located features in a SPM, a feature text description and the number of the model was given. To locate features in the PMF only the feature text description was given to the subjects, and they had to find the model elements of the feature in the collection of elements of five product models.

Table 3 shows the characteristics of the features used in the experiment in terms of the size and the percentage of model elements from the feature in the collection of elements of the SPM or the PMF. Figure 5 shows the realization of the feature described by the text 'High power external supply for induction chain' (F08 Table 3).Since it is composed of three elements, its size is three. The percentage of model elements from the feature increases when the FL focuses on one single product model and also when the number of model elements in the feature increases. The model elements of the features 03, 07, and 11 appear in two different product models when the feature is located in the Product Model Family. The table includes information about the model

Table 3: Description of experimental objects

		FL Scope			
		SPM		**PMF**	
	$Size\,(F)$	$\frac{Size(F)}{Size(PM_i)}\%$	PM_i	$\frac{Size(F)}{Size(PMF)}\%$	$\lvert MF\rvert$
F01	1	6%	PM_1	1%	1
F02	2	11%	PM_1	2%	1
F03	3	13%	PM_3	4%	2
F04	8	53%	PM_5	10%	1
F05	2	13%	PM_5	2%	1
F06	2	11%	PM_1	2%	1
F07	2	7%	PM_2	2%	2
F08	3	16%	PM_4	4%	1
F09	2	13%	PM_5	2%	1
F10	4	21%	PM_4	5%	1
F11	2	7%	PM_2	2%	2
F12	1	5%	PM_4	1%	1

used when FL is in SPM and the number of product models containing the feature ($|MF|$) when FL is in a PMF.

Each solution output by the subjects is a subset of model elements in the *FL Scope* where the feature is being located. To describe performance, the confusion matrix classifies the solutions output by the subjects [33, 19].

The confusion matrix distinguishes between two values: true or present and false or absent. The predicted values (model elements in the subject's solution) and the model elements of the feature to be located are compared, and the confusion matrix arranges the results of the comparison into four categories. These are: True Positive (TP), predicted values that are in the feature to be located; False Positive (FP), predicted values that are not in the feature to be located; True Negative (TN), model elements that are neither predicted values nor elements in the feature to be located; and False Negative (FN), elements in the feature to be located that are not predicted values.

Recall, Precision, and F-measure are defined from these predicted values of the confusion matrix. Recall is the ratio between TP and TP+FN. It represents the percentage of the feature elements that appear in the subject's solution. Precision is the ratio between TP and (TP+FP). It represents the percentage of the elements in the subject's solution that are in the set that represents the feature. The F-measure is the harmonic mean of Recall and Precision: $F - measure = \left(\frac{(Recall)^{-1}+(Precision)^{-1}}{2}\right)^{-1}$.

The feature realizations of the Family Product Model are known beforehand. For each feature text description given to the subjects, the set of product model elements that are implemented in the feature is known. It is the ground truth (the oracle) that allows us to build the confusion matrix for the subjects' solutions when locating features.

With the following example, we illustrate the potential of Recall, Precision, and F-measure values to describe performance when locating features and how the F-measure could be interpreted as the percentage of problem solved by a subject. Figure 5 shows the graphical representation of the realization of the feature (F) described by the text 'High power external supply for induction chain' in a Product Model. Let us assume that the solutions provided by three subjects (A, B, and C) for the feature location of the feature F, were $S_A = \{10, 11\}$, $S_B = \{10, 11, 12, 14, 15\}$, and $S_C = \{9, 3, 11\}$, respectively. The real value of the feature given by the oracle is $F = \{10, 11, 12\}$.

Table 4: Example of performance measurements

	TP	FP	FN	Precision	Recall	F-Measure
Oracle	3	0	0	100%	100%	100%
S_A	2	0	1	100%	67%	80%
S_B	2	3	1	40%	67%	50%
S_C	1	2	2	33%	33%	33%

Table 4 shows the values of TP, FP, and FN in the confusion matrix and the correspondent values of Recall, Precision, and F-measure for the oracle and for the solutions S_A, S_B, and S_C. The solution of A (S_A) has a precision of 100%, but its recall value is 67% since it does not contain all of the model elements of the oracle. For the solution S_A, the F-measure is 80%. The recall value for the solution of B (S_B) is also 67%, but the precision value is only 40% which is the percentage of elements of this solution that are in the oracle. The F-measure of S_B is only 50%. The solution S_C obtains values of 33% for Precision, Recall, and F-measure since the solution only contains one element of the oracle and it contains two elements that are not in the oracle.

5.4. Variables

The independent variable in this study is the *FL Scope*. It has two values, Single Product Model (SPM) and Product Model Family (PMF), which are two different kinds of sets of model elements used by subjects to solve the FL tasks.

We consider three dependent variables: *Performance*, *Productivity*, and *Perceived Difficulty*.

To measure *Performance*, we built the confusion matrix of each one of the six features that the subjects must locate for each task to calculate its recall, precision, and F- measure. To represent the *Performance* of the subjects in each task, we calculated the arithmetic mean of the F-measures obtained for each one of the six features to be located in the task.

We measured the time used by each subject to finish each task to calculate *Productivity* as the ratio of *Performance* to time spent (in minutes) to perform a task.

At the end of each task, each subject rated the level of difficulty of the task using a 7-point Likert-scale, ranging from *too easy* to *very difficult*. This value was used to measure *Perceived Difficulty*.

5.5. Design

In order to improve experiment robustness regarding variation among subjects [11], we chose a repeated measurement using the largest possible sample size. To avoid the order effect, we chose a crossover design, and we grouped the features to be located in two different tasks, T1 and T2. The subjects had to locate six different features in each one of the tasks. All of the subjects located six features in a Single Product Model in one task, and they located another six different features in a Product Model Family in the other task.

The subjects had been randomly divided into two groups (G1 and G2). In the first part of the experiment, all of the subjects solved T1, G1 located features in the Product Model Family, and G2 located the same features in a Single Product Model. Afterwards, all of the subjects solved T2, G1 located features in a single Product Model, and G2 located features in a Product Model Family. Table 5 shows a summary of the experiment crossover design.

Table 5: Experiment Design

	Task	
Group	*Task 1*	*Task 2*
G1	Product Model Family	Single Product Model
G2	Single Product Model	Product Model Family

The two tasks were designed by the same instructor. Both were similar yet independent enough to avoid the learning effect. To verify the experiment design, we conducted a pilot study with one subject. This pilot study allowed us to detect typographical and semantic mistakes in some expressions, which were corrected for the experiment. The subject in the pilot study did not participate in the experiment described above.

This design avoids the learning problem effect and the variability due to differences in the average response capacity of the subjects.

5.6. Participants

The subjects were selected according to convenience sampling [32]. A total of 18 subjects performed the experiment. All of the subjects completed a demographic questionnaire before entering the experiment, which was used to characterize the sample. Five subjects were software developers and researchers in the induction hob domain. They stated spending an average of 4 hours per day developing software, and an average of 4.2 hours per day working with modeling languages. They were the experts in the experiment. There were also 13 Master's students from Universidad San Jorge (Zaragoza, Spain). They were not experts in the induction hob domain. They stated spending an average of 4.9 hours per day developing software, and 1.34 hours per day working with modeling languages. They were the non-experts in the experiment.

The experiment was conducted by two instructors. The instructors are senior software engineers from the company, and they are also responsible for training newcomer engineers after a hiring process. One instructor designed the tasks for the experiment and generated the correction templates. This instructor clarified general doubts during the the experiment and took notes during the focus group. The other instructor explained the experiment, provided information, clarified doubts about the DSL used in the experiment, managed the focus groups, and corrected the tasks.

5.7. Experiment procedure

The diagram in Figure 6 summarizes the experimental procedure, which is described as follows:

1. The subjects received information about the experiment. An instructor explained the parts in the session. He advised that it was not a test of their abilities.

2. The subjects attended a tutorial about FL in software models, and about the DSL used in the experiment. The information used in the tutorial was available to the subjects during the experiment. The average time spent on this tutorial was 15 minutes.
3. The subjects completed a demographic questionnaire. One of the instructors distributed and collected the questionnaires, verifying that all of the fields had been answered and that the subject had signed the voluntary participation form in the experiment.
4. The subjects received clear instructions on where to find the statements for each task, how to submit their work, and how to complete the task sheet at the end of each task.
5. The subjects performed the first task. The subjects were randomly divided into two groups (G1 and G2) to locate features from the first task. The subjects from G1 had to locate six features in the Product Model Family, and the subjects from G2 had to locate the same six features in a Single Product Model.
6. The subjects assessed the difficulty of the task, taking into account where the feature was located, in a Single Product Model or in the Product Model Family.
7. The subjects added comments about the process followed to locate the feature elements of the first task.
8. An instructor checked that each subject had filled in all of the fields on the task sheet.
9. The subjects performed the second task. The subjects from G1 located six features in a Single Product Model, and the subjects from G2 located the same six features in the Product Model Family. Then, the subjects assessed the difficulty of the second task and added comments about the process followed to locate the feature elements of the second task.
10. A focus group interview about the tasks was conducted by one instructor, while the other instructor took notes.
11. Finally, an instructor corrected the tasks, and a researcher analyzed the results.

The experiment was conducted on two different days at Universidad San Jorge (Zaragoza, Spain). On the first day, it was performed by a group of thirteen Master's students (non-experts) in a subject about advanced software modeling. On the second day, the same experiment was performed by

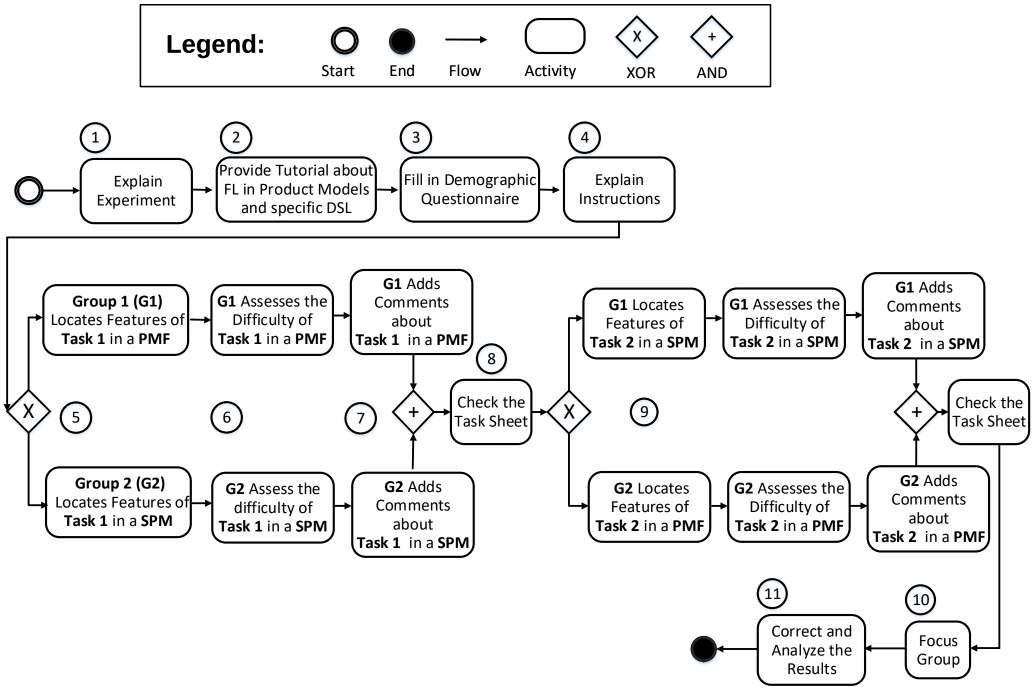
Legend:
Start
End
Flow
Activity
X
XOR
+
AND
1
Explain Experiment
2
Provide Tutorial about FL in Product Models and specific DSL
3
Fill in Demographic Questionnaire
4
Explain Instructions
X
5
Group 1 (G1) Locates Features of Task 1 in a PMF
Group 2 (G2) Locates Features of Task 1 in a SPM
6
G1 Assesses the Difficulty of Task 1 in a PMF
G2 Assess the difficulty of Task 1 in a SPM
7
G1 Adds Comments about Task 1 in a PMF
G2 Adds Comments about Task 1 in a SPM
+
8
Check the Task Sheet
X
9
G1 Locates Features of Task 2 in a SPM
G2 Locates Features of Task 2 in a PMF
G1 Assesses the Difficulty of Task 2 in a SPM
G2 Assesses the Difficulty of Task 2 in a PMF
G1 Adds Comments about Task 2 in a SPM
G2 Adds Comments about Task 2 in a PMF
+
Check the Task Sheet
10
Focus Group
11
Correct and Analyze the Results

Figure 6: Experimental Procedure

the five experts divided into two groups based on their schedule availability (Resp., 2experts, 3 experts).

The instructor responsible for designing the tasks was not the same one who explain the tutorial about FL in software models and the DSL used in the experiment. The materials used in this experiment, which includes the training material, the consent to process the data, the demographic questionnaire, the task sheets, and the results are available at `http://svit.usj.es/ManualFL-experiment`.

5.8. Analysis procedure

For the data analysis, we have chosen the Linear Mixed Model (LMM) [34]. LMM handles correlated data resulting from repeated measurements. It is one of the analysis method recommended for crossover experiments in software engineering [35]. The dependent variables for this test are *Performance*, *Productivity*, and *Perceived Difficulty*.

In this study, the subjects who participated are consider random factors and the *FL Scope* is the variable that is repeated to identify the differences between locating features in a Single Product Model or in a Product Model Family. The experimental factors, which are considered fixed effects in our statistical model, are the *FL Scope* (FLS), the *Experience* (E), and the sequence *FL Scope* and *Experience* ($FLS * E$). The subjects are the random effects. The statistical model is expressed in the following formula, where DV represents the dependent variable:

$$DV \sim \quad FLS + E + FLS * E + (1|\,Subject) \tag{1}$$

Since the primary focus in our investigation is the *FL Scope*, it is a fixed effect. However, we add *Experience* and the sequence *FL Scope* and *Experience* as be fixed effects for their potential in determining the variability in the dependent variables due to *FL Scope*. There are studies on manual feature location [14, 3] which affirm that human factors such as experience affect feature location patterns and actions, that are improving the effectiveness of the FL. Adding fixed factors related to the subject's experience in the statistical model improves the quality of the model in order to explain the variance of the dependent variables.

To quantify the difference between FL in a Single Product Model and FL in a Product Model Family, we have calculated the effect size using the means and the standard deviations of the values of *Performance*, *Productivity*, and *Perceived Difficulty* for each one of the *FL Scopes* where the subjects locate features. With these values, we calculated Cohen's d Value [36], which is the standardized difference between the two means. Values of Cohen d between 0.2 and 0.3 indicate a small effect, values around 0.5 indicate a medium effect, and values greater than 0.8 indicate a large effect. This value also allows us to measure the percentage of overlap, the percenta between the distributions of the dependent variables for FL in a Single Product Model and the distributions of the dependent variables for FL in a Product Model Family.

6. Results

Table 6 shows the values for the mean of the dependent variables *Performance*, *Productivity*, and *Perceived Difficulty* for each one of the *FL Scopes* in which the subjects locate features: Single Product Model and Product Model Family.

There are differences in the means of all the dependent variables depending on which *FL Scope* was used to locate features. These differences are

Table 6: Values for the mean of the dependent variables

		Performance	*Productivity*	*Perceived*
		%	*%/min*	*Difficulty*
Experts	SPM	65%	8.3%/min	2.2
	PMF	47%	3.6%/min	3.0
Non-experts	SPM	35%	3.9%/min	3.0
	PMF	23%	2.2%/min	3.3
All subjects	SPM	43%	5.1%/min	2.8
	PMF	29%	2.6%/min	3.2

found in both, the experts and the non-experts. The hypothesis testing will allow us to confirm whether or not these differences are significant.

6.1. Hypothesis testing

The use of the Linear Mixed Model test assumed that dependent variables residuals must be normally distributed. The normality of these residuals had been verified by the Shapiro-Wilk and Kolmogorov-Smirnov tests, in addition to visual inspections of the histogram and normal Q-Q plots. We obtained normally distributed residuals for all of the dependent variables. All of the residuals obtained a p-value greater than 0.05 with the normality tests.

The results of the Type III test of fixed effects for all of the dependent variables are shown in Table 7.

Table 7: Results of fixed effects for each variable

			Perceived
	Performance	*Productivity*	*Difficulty*
FLS	(F=12.0, p=.003)	(F=10.3, p=.003)	(F=3.0, p=.102)
E	(F=15.7, p=.001)	(F=8.2, p=.007)	(F=1.1, p=.307)
$FLS * E$	(F=.5, p=.490)	(F=2.3, p=.143)	(F=.6, p=.452)

For the variables *Performance* and *Productivity*, the factor *FL Scope* obtained p-values of less than 0.05. Therefore, our first two null hypotheses are rejected. Thus, the answers to research questions **RQ1** and **RQ2** are affirmative. The *FL Scope* used for locating features has a significant impact on *Performance* and *Productivity*.

However, since *FL Scope* obtained p-values greater than 0.05 for the variable *Perceived Difficulty*, we can not reject our third null hypothesis . Thus, the answer to research question **RQ3** is negative. The *FL Scope* used for locating features does not have significant impact on the difficulty perceived by the subjects.

The factor *Experience* obtained p-values of less than 0.05 for *Performance* and *Productivity* but greater than 0.05 for *Perceived Difficulty.* This factor also explains the changes in the first two dependent variables, but not in *Perceived Difficulty.*

Moreover, the fixed factor *FL Scope*Experience* obtained p-values greater than 0.05 for all of the dependent variables, which implies that the combination of *FL Scope* and *Experience* had no significant influence on the changes in *Performance*, *Productivity*, or *Perceived Difficulty.* By changing the *FL Scope*, the effects on the dependent variables occur in the same direction and with a similar intensity for both groups of subjects (experts and non-experts).

6.2. Effect size

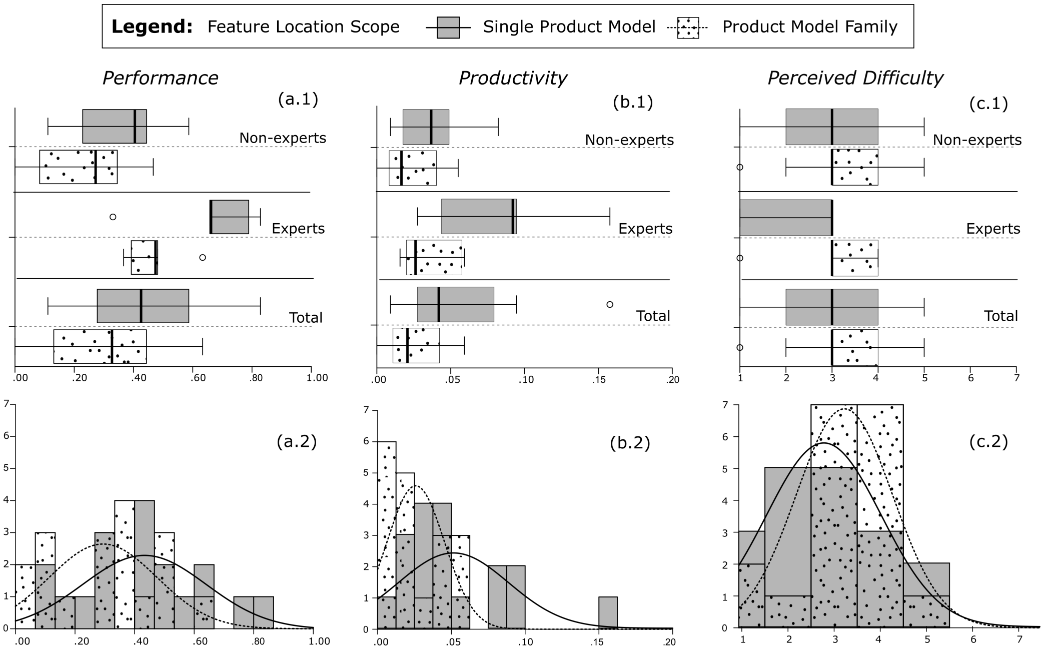

Figure 7: Box plots and histograms with normal distribution for *Performance* ((a.1) and (a.2), respectively); for *Productivity* ((b.1) and (b.2)); and for *Perceived Difficulty* ((c.1) and (c.2))

The effect size of the differences based on manual FL in a Single Product Model or in a Product Model Family for *Performance* is medium-large, with a Cohen d value of 0.703. The effect size for *Productivity* is large, with a Cohen d value of 0.847. These values indicate that the percentage of overlap between the distribution of the variables, when the FL is in a Single Product Model and when the FL is in a Family Product, is less than 60%. Taking into account the area covered by each of the two distributions, the percentage of area shared by both distributions is less than 60%. The percentage of area covered by one of the distributions, but not the other, is more than 40% (percentage of non-overlap)

The distribution of *Performance* when the FL is in a Single Product Model with the distribution of *Performance* when the FL is in a Family Product model is more than 43%. The percentage of non-overlap of *Productivity* distributions for FL in a Single Model and FL in a Family Product Model is more than 47%. The box plots and the histograms of Figure 7 illustrate the differences in *Performance* ((a.1) and (a.2)) and in *Productivity* ((b.1) and (b.2)). In histograms of Figure 7, the non-overlapping parts have a single pattern (either dots or shaded), while the overlapping parts have both patterns (dots and shaded).

The box plots and the histograms of Figure 7 for *Perceived Difficulty*,(c.1) and (c.2), respectively, illustrate that the difference when the *FL Scope* changes is not larges. A Cohen d value of -0.381 suggested a moderate effect in favour of FL in a Product Model Family. The subjects perceived slightly more difficult FL in a Product Model Family than in a Single Product Model. The percentage of non-overlap of *Perceived Difficulty* distributions for FL in a Single Model and FL in a Family Product Model is less than 29%. Its histogram has about 70% overlapping.

This data allows us to give more precise answers to **RQ1**, **RQ2**, and **RQ3**. For *Performance* and *Productivity*, the impact of the *FL Scope* used for locating features is medium-large, and large and the difference is statistically significant. On contrast, for *Perceived Difficulty*, the magnitude of the difference is moderate and is not significant.

7. Discussion

To better explain the results, we analyzed the performance values for each feature. For 89% of the features, 100% of Precision, Recall, and F-measure was not reached (87% for SPM, 92% for PMF). All of the subjects had errors in some of the features they located. Only two subjects (experts) found elements for all of the features to be located in a SPM. No subject located the elements for all of the features when they located features in the PMF.

We analyzed the confusion matrix to measure the subjects' errors in each feature when the 100% performance was not reached. False Positive was the error that most frequently appeared, and the number of errors increased when FL was in a PMF. When FL was in a PMF, there were 25% more False Positives. Some subjects acknowledged that after locating feature elements, they reviewed the other product models looking for these elements. When the subjects select a non-feature element, if that element is shared by other product models in the family, then the error is propagated. We found that 8% of the subjects' solutions (9 of 108) were affected by fault propagation. This fault propagation was detected in the solutions of four non-experts (30% of the non-experts). The solutions of the experts were not affected by fault propagation.

PMF also brings another type of error to the table. When the FL was in a PMF, there were 67% more False Negative elements than when FL was in a SPM. Both the experts and non-experts described a process of elimination to locate features. The elimination process was not taught in the training session. One expert and one non-expert explicitly described an elimination process on their task sheets. Two experts and three non-experts stated that they used something that was unique in the feature description as a filter to discard elements or models during the focus group. As the number of elements of the FL scope increases, the more difficult it is to find the correct feature elements using a process of elimination.

When the subjects were locating features in the PMF, they did not always select the correct product model where the feature elements were located. Some subjects indicated that the feature was not in any product model. On average, the experts selected the product model where the feature elements were located for 60% of the features to be located. The non-experts selected the correct product model for the 50% of the features, on average. The only time that all the subjects selected the correct product model was for the feature F04. This is the largest feature in the set of features. This feature also had the best performance: 85% when the feature is beeing located in a SPM, and 78% when the feature is locating in a PMF. Performance is also related to the number of feature elements; features with more elements obtained better performance.

We also analyzed the changes in performance based on other characteristics of the features to be located. We used the five measurements (Size, volume, density, multiplicity, and dispersion) proposed by Ballarín et al. [22]. We found that dispersion was the measurement that best explains the difference in performance when the FL scope changes, based on the characteristics of the location problem. The dispersion measures the ratio of connected elements in the feature. If a feature is composed by three elements but only two of them are connected it is considered that the feature has two groups of elements. Dispersion is computed as the ratio between the number of groups and the number of elements of a feature [22]. For example, if a feature is composed of three elements in two groups, the dispersion is 2/3.

Table 8: Performance Measures by Dispersion

Dispersion Range	**Precision**	**Recall**	**F-measure**	**FL Scope**
0-0.3	65%	70%	63%	SPM
	64%	56%	57%	PMF
0.5-0.7	56%	61%	55%	SPM
	34%	40%	35%	PMF
1	12%	44%	36%	SPM
	6%	18%	9%	PMF

We classified the features using dispersion. In our experiment, there are two features with a dispersion values of less than 0.3, six features with dispersion values between 0.5 and 0.7, and four features with a dispersion value of one. Table 8 shows the average values of performance for the features of each range. As can be observed, the stronger the dispersion, the lower the performance and the greater the differences in performance between locating features in a SPM and locating features in a PMF.

Table 9 summarizes the results, the answers, and the findings related to each research question. The above findings (the largest features, feature dispersion, feature propagation, and the elimination process) can help the research community to develop better approaches in the context of feature location.

- **Largest features**: We recommend that the scientific community conduct experiments to study the influence of feature size on FL tasks in detail. To date, no related work has identified the influence of this factor on the performance of FL tasks. The feature size may have the potential to help determine when to use manual, automated, or semi-automated FL. So far, there are no concrete recommendations on

Table 9: Summary of Results, Answers, and Findings by Research Questions

RQ1	**Results**	For *Performance*, the factor *FL Scope* obtained a p-value of less than 0.05 in the hypothesis contrast test; therefore, the null hypothesis of equal distributions is rejected. The effect size for *Performance* is medium-large in favour of FL in a SPM, with a Cohen d value of 0.703.
	Answer	The *FL Scope* used for locating features has a significant impact on *Performance*. The *Performance* is worse when FL is in a PMF, with the impact being medium-large.
	Findings	· The number of errors increased when FL was in a PMF. · When FL is in a PMF, error propagation affects 30% of non-experts but does not affect experts. · When subjects locate features in PMF, they can fail in selecting the product model where the feature is actually located. · The process of elimination as a FL Technique works worse in PMF than in SPM. · Features with a larger size (more elements) get better performance in SMP and in PMF. · Dispersion was the measurement that best explains the difference in performance when the *FL Scope* changes. · The stronger the dispersion, the lower the performance and the greater the differences between locating features in a SPM and locating features in a PMF.
RQ2	**Results**	For *Productivity*, the factor *FL Scope* obtained a p-value of less than 0.05 in the hypothesis contrast test; therefore, the null hypothesis of equal distributions is rejected. The effect size for *Productivity* is large in favour of SPM, with a Cohen d value of 0.847.
	Answer	The *FL Scope* used for locating features has a significant impact on *Productivity*. The *Productivity* is worse when FL is in a PMF, with the impact being large.
	Findings	· The initial inclination of subjects is to think that locating features in a PMF only takes more time than locating features in a SPM. · Features with a larger size (more elements) get better productivity in SMP and in PMF. · Dispersion was the measurement that best explains the difference in productivity when the *FL Scope* changes. · The stronger the dispersion, the lower the productivity and the greater the differences between locating features in a SPM and locating features in a PMF.
RQ3	**Results**	For *Perceived Difficulty*, the factor *FL Scope* obtained a p-value of more than 0.05 in the hypothesis contrast test; therefore, the null hypothesis of equal distributions is not rejected. The effect size of the differences for *Perceived Difficulty* is moderate in favour of FL in a PMF, with a Cohen d value of -0.381.
	Answer	*FL Scope* used for locating features does not have a significant impact on *Perceived Difficulty*.
	Findings	· The subjects (experts and non-experts) underestimate the difficulty of FL in a Poduct Model Family. · The difficulty that subjects perceive does not predict the significant worsening of the results in performance and productivity. · The subjects are not aware of the tricky challenge that FL presents in SPL reengineering.

when to use each of these approaches. Engineers may think that it is not worth using automated FL to locate a small feature, so they locate it manually. This idea can backfire, and according to our findings, the size of the feature could help engineers to select the best approaches.

- **Feature propagation**: As Krüger et al. [25] stated, many automated and semi-automated FL techniques require a seed for the feature to be located, and this seed must be located manually. When these seeds are obtained by manual FL, they may contain wrong elements and this error could be propagated, as we have shown in our work. Acknowledging this weakness is the first step in designing input filters that improve the quality of the seeds.

- **Elimination process**: Automated and semi-automated FL techniques explore different approaches to guide feature location (e.g., latent semantic analysis [19] or empirical learning [17]). These techniques search for the relevant elements of the feature. However, the process of elimination could provide inspiration to complement these techniques. For example, these techniques could add a phase in which the model elements were scored based on their differences with the description of a certain feature. The implementation of this process of elimination in FL techniques is out of the scope of this work; however, we want to emphasize that this explicit process of elimination could be a new direction in research to improve FL techniques.

- **Feature dispersion**: This factor, like the ones above, can also contribute to defining feature profiles to recommend which FL approach to use. Therefore, we recommend that experiments delve into how this factor can influence the performance of FL tasks. This factor may also be relevant in rethinking the way in which the results of feature location techniques are presented to engineers. Most of the FL techniques use feature rankings in order to present the results by relevance. Since the dispersion is related to the performance of the engineers, the rankings should offer the possibility of also presenting the results by dispersion ranges and groups of elements. This more complete information could help engineers decide which result in the feature ranking should be chosen.

Furthermore, understanding the difficulties in manual FL is also helpful in order to not miss relevant test cases when FL is evaluated. Our results identify cases where the performance of the engineers in this respect is worse, and these cases can be used as test-cases to evaluate whether an automated or a semi-automated technique could compensate for this.

With regard to generalization, the design of our experiment not only considers the specific characteristics of the domain used by our industrial partner. In fact, it could be applied to any domain that uses the widespread MOF modeling standard. This is why the tasks and the results are expressed in terms of model elements (classes, references, and properties) and not in terms of domain-dependent concepts (inverter, power channel, power manager, or inductor). For example, the five measurements (size, volume, density, multiplicity, and dispersion) that we have used in the discussion were defined in another work [22] using examples from a completely different domain (train control software) and are applicable to any domain.

8. Threats to validity

To describe the threats of validity of our work, we use the classification of [32]:

Conclusion validity. The *low statistical power* was minimized because the confidence interval is 95%. To minimize the *fishing and the error rate* threat, the tasks and corrections were designed by an instructor, with experience in the DSL used, who did not participate in the correction process. The *Reliability of measures* threat was mitigated because two instructors tested the data coherence of the subjects task sheets at the end of each task. The *reliability of treatment implementation* threat was alleviated because the treatment implementation was identical in the two sessions.

Internal validity. To avoid the *instrumentation* threat, we conducted a pilot study to verify the design and the instrumentation. The *interactions with selection* threat affected the experiment because there were subjects who had different levels of experience. To mitigate this threat, the treatment was applied randomly.

Construct validity. *Mono-method bias* occurs due to the use of a single type of measure [37]. All of the measurements were affected by this threat. To mitigate this threat, an instructor checked that the subjects performed the tasks, and we mechanized these measurements as much as possible by means of correction templates. The *hypothesis guessing* threat appears when the subject thinks about the objective and the results of the experiment. To mitigate this threat, we did not explain the research questions or the experiment design to the subjects. The *evaluation apprehension* threat appears when the subjects are afraid of being evaluated. To weaken this threat, at the beginning of the experiment, the instructor explained to the subjects that the experiment was not a test about their abilities. *Author bias* occurs when the people involved in the process of creating the experiment artifacts subjectively influence the results. In order to mitigate this threat, the tasks were balanced, i.e., their sizes and difficulty were the same for the two tasks. Finally, the *mono-operation bias* threat occurs when the treatments depend on a single operationalization. The experiment was affected by this threat since we worked with a specific DSL.

External validity. The *interaction of selection and treatment* threat is an effect of having a subject that is not representative of the population that we want to generalize.The experiment was performed by non-experts and experts. The participation of non-experts can be a source of experiment weakness; nevertheless, using students instead of software engineers is not a major issue as long as the research questions are not specifically focused on experts [38, 39]. Our results are similar to those of other studies in which the performance of experts and students is compared when locating features in code: experts perform better than non-experts on FL tasks [13, 3, 14]. In our experiment, we also found that regardless of experience, when the scope changes and the FL tasks are performed on a family of products, the performance and productivity worsens. The performance and productivity of experts and non-experts is equally impaired when FL tasks are performed in a product family. The *interaction of setting and treatment* threat is an effect of not having material that is representative of the industrial context of study. The domain used in the experiment is sufficient to perform FL tasks that are analogous to those commonly performed by the engineers of our industrial partner. The values achieved for performance show that all of the subjects had errors in some of the features they located. Only two subjects (experts) found elements for all of the features to be located and only in a single product model. These facts suggest that the tasks are non-trivial. Furthermore, the domain used in this experiment has already been used in previous works [24][40]. The *domain* threat appears since we only analyzed the induction hob 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.

9. Conclusion

In this work, we present an experiment that compares performance, productivity, and perceived difficulty in manual FL when the scope changes from a single product to a product family. The experimental objects are extracted from a real-world SPL that uses a DSL. While the performance and the productivity decreases significantly when engineers locate features in a product family, the perceived difficulty does not have very significant changes.

A more thorough analysis seems to indicate how the feature size or the feature dispersion can explain the changes in performance when the scope changes. These results suggest a new research direction that can increase the understanding of the feature problem in order to take advantage of strengths and compensate for weaknesses when developing or evaluating new approaches to FL.

In this work the subjects located features in isolation, but in feature-based product lines when features are added to a product model, interactions may appear between the features of that product or the features of other products in the family. Interactions can be severely damaging to system development and to user expectations [41]. The analysis of interactions between features could also be used to determine the original feature module definitions [42, 43]. None of the authors referenced in the Related Work section has explicitly analyzed the effects of feature interaction on the performance of FL tasks. With the aim of understanding the feature problem in an SPL, feature interactions must also be considered, which is what we plan to do in future works.

References

[1] J. Krüger, W. Gu, H. Shen, M. Mukelabai, R. Hebig, T. Berger, Towards a better understanding of software features and their characteristics: a case study of marlin, in: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems, 2018, pp. 105–112.

[2] W. Ji, T. Berger, M. Antkiewicz, K. Czarnecki, Maintaining feature traceability with embedded annotations, in: Proceedings of the 19th International Conference on Software Product Line, 2015, pp. 61–70.

[3] J. Wang, X. Peng, Z. Xing, W. Zhao, How developers perform feature location tasks: a human-centric and process-oriented exploratory study, Journal of Software: Evolution and Process 25 (11) (2013) 1193–1224.

[4] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, V. Rajlich, Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval, IEEE Transactions on Software Engineering 33 (6) (2007) 420–432.

[5] T. J. Biggerstaff, B. G. Mitbander, D. Webster, The concept assignment problem in program understanding, in: [1993] Proceedings Working Conference on Reverse Engineering, IEEE, 1993, pp. 27–43.

[6] A. J. Ko, B. A. Myers, M. J. Coblenz, H. H. Aung, An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks, IEEE Transactions on software engineering 32 (12) (2006) 971–987.

[7] W. K. Assunção, S. R. Vergilio, R. E. Lopez-Herrejon, Automatic extraction of product line architecture and feature models from uml class diagram variants, Information and Software Technology 117 (2020) 106198.

[8] J. Martinez, T. Ziadi, T. F. Bissyande, J. Klein, Y. Le Traon, Automating the extraction of model-based software product lines from model variants (t), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2015, pp. 396–406.

[9] K. Pohl, G. Böckle, F. J. van Der Linden, Software product line engineering: foundations, principles and techniques, Springer Science & Business Media, 2005.

[10] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, A. Wasowski, A survey of variability modeling in industrial practice, in: Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, 2013, pp. 1–8.

[11] N. Wilde, M. Buckellew, H. Page, V. Rajlich, L. Pounds, A comparison of methods for locating features in legacy software, Journal of Systems and Software 65 (2) (2003) 105–114.

[12] M. Revelle, T. Broadbent, D. Coppit, Understanding concerns in software: insights gained from two case studies, in: 13th International Workshop on Program Comprehension (IWPC'05), IEEE, 2005, pp. 23–32.

[13] J. Wang, X. Peng, Z. Xing, W. Zhao, An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions, in: 2011 27th IEEE International Conference on Software Maintenance (ICSM), IEEE, 2011, pp. 213–222.

[14] H. Jordan, J. Rosik, S. Herold, G. Botterweck, J. Buckley, Manually locating features in industrial source code: the search actions of software nomads, in: 2015 IEEE 23rd International Conference on Program Comprehension, IEEE, 2015, pp. 174–177.

[15] K. Damevski, D. Shepherd, L. Pollock, A field study of how developers locate features in source code, Empirical Software Engineering 21 (2) (2016) 724–747.

[16] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Feature location in models through a genetic algorithm driven by information retrieval techniques, in: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, 2016, pp. 272–282.

[17] A. C. Marcén, J. Font, Ó. Pastor, C. Cetina, Towards feature location in models through a learning to rank approach, in: Proceedings of the 21st International Systems and Software Product Line Conference-Volume B, 2017, pp. 57–64.

[18] F. Pérez, R. Lapeña, J. Font, C. Cetina, Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study, Information and Software Technology 103 (2018) 188–201.

[19] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Achieving feature location in families of models through the use of search-based software engineering, IEEE Transactions on Evolutionary Computation 22 (3) (2017) 363–377.

[20] L. Arcega, J. Font, Ø. Haugen, C. Cetina, Leveraging Models at Run-Time to Retrieve Information for Feature Location., in: MoDELS@ Run. time, 2015, pp. 51–60.

[21] C. Cetina, J. Font, L. Arcega, F. Pérez, Improving feature location in long-living model-based product families designed with sustainability goals, Journal of Systems and Software 134 (2017) 261–278.

[22] M. Ballarín, A. C. Marcén, V. Pelechano, C. Cetina, Measures to report the location problem of model fragment location, in: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, 2018, pp. 189–199.

[23] F. Pérez, J. Font, L. Arcega, C. Cetina, Collaborative feature location in models through automatic query expansion, Automated Software Engineering 26 (1) (2019) 161–202.

[24] F. Pérez, J. Echeverría, R. Lapeña, C. Cetina, Comparing manual and automated feature location in conceptual models: A controlled experiment, Information and Software Technology 125 (2020) 106337.

[25] J. Krüger, T. Berger, T. Leich, Features and how to find them: A survey of manual feature location, Software Engineering for Variability Intensive Systems (2019) 153–172.

[26] A. Lai, G. C. Murphy, The structure of features in java code: An exploratory investigation, in: Position Paper for Multi-Dimensional Separation of Concerns Workshop, OOPSLA, 1999.

[27] J. Starke, C. Luce, J. Sillito, Searching and skimming: An exploratory study, in: 2009 IEEE International Conference on Software Maintenance, IEEE, 2009, pp. 157–166.

[28] J. Rubin, M. Chechik, A survey of feature location techniques, in: Domain Engineering, Springer, 2013, pp. 29–58.

[29] O. Pastor, J. C. Molina, Model-driven architecture in practice: a software production environment based on conceptual modeling, Springer Science & Business Media, 2007.

[30] D. Blasco, J. Font, M. Zamorano, C. Cetina, An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering, Journal of Systems and Software 171 (2021) 110804.

[31] O.M.G.: Meta object facility (mof) version 2.4.1, http://www.omg.org/spec/MOF/2.4.1/ (2013).

[32] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.

[33] S. V. Stehman, Selecting and interpreting measures of thematic classification accuracy, Remote sensing of Environment 62 (1) (1997) 77–89.

[34] B. T. West, K. B. Welch, A. T. Galecki, Linear mixed models: a practical guide using statistical software, Chapman and Hall/CRC, 2014.

[35] S. Vegas, C. Apa, N. Juristo, Crossover designs in software engineering experiments: Benefits and perils, IEEE Transactions on Software Engineering 42 (2) (2015) 120–135.

[36] J. Cohen, Statistical power for the social sciences, Hillsdale, NJ: Laurence Erlbaum and Associates (1988).

[37] J. I. Panach, S. España, Ó. Dieste, Ó. Pastor, N. Juristo, In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction, Information and Software Technology (2015).

[38] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Transactions on Software Engineering 28 (8) (2002) 721–734.

[39] I. Reinhartz-Berger, D. Dori, Opm vs. uml–experimenting with comprehension and construction of web application models, Empirical Software Engineering 10 (1) (2005) 57–80.

[40] J. Echeverría, F. Pérez, J. I. Panach, C. Cetina, An empirical study of performance using clone & own and software product lines in an industrial context, Information and Software Technology 130 (2021) 106444.

[41] M. Calder, M. Kolberg, E. H. Magill, S. Reiff-Marganiec, Feature interaction: a critical review and considered forecast, Computer Networks 41 (1) (2003) 115–141.

[42] D. Batory, P. Höfner, J. Kim, Feature interactions, products, and composition, in: Proceedings of the 10th ACM international conference on Generative programming and component engineering, 2011, pp. 13–22.

[43] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, G. Saake, Predicting performance via automated

feature-interaction detection, in: 2012 34th International Conference on Software Engineering (ICSE), IEEE, 2012, pp. 167–177.