Comparing Manual And Automated Feature Location In Conceptual Models: A Controlled Experiment

Francisca Pérez*, Jorge Echeverría, Raúl Lapeña, Carlos Cetina

Universidad San Jorge. SVIT Research Group
Autovía A-23 Zaragoza-Huesca Km.299, 50830, Zaragoza, Spain

Abstract

Context: Maintenance activities cannot be completed without locating the set of software artifacts that realize a particular feature of a software system. Manual Feature Location (FL) is widely used in industry, but it becomes challenging (time-consuming and error prone) in large software repositories. To reduce manual efforts, automated FL techniques have been proposed. Research efforts in FL tend to make comparisons between automated FL techniques, ignoring manual FL techniques. Moreover, existing research puts the focus on code, neglecting other artifacts such as models.

Objective: This paper aims to compare manual FL against automated FL in models to answer important questions about performance, productivity, and satisfaction of both treatments.

Method: We run an experiment for comparing manual and automated FL on a set of 18 subjects (5 experts and 13 non-experts) in the domain of our industrial partner, BSH, manufacturer of induction hobs for more than 15 years. We measure performance (recall, precision, and F-measure), productivity (ratio between F-measure and spent time), and satisfaction (perceived ease of use, perceived usefulness, and intention to use) of both treatments, and perform statistical tests to assess whether the obtained differences are significant.

Results: Regarding performance, manual FL significantly outperforms automated FL in precision and F-measure (up to 27.79% and 19.05%, respectively), whereas automated FL significantly outperforms manual FL in recall (up to 32.18%). Regarding productivity, manual FL obtains 3.43%/min, which improves automated FL significantly. Finally, there are no significant differences in satisfaction for both treatments.

Conclusions: The findings of our work can be leveraged to advance research to improve the results of manual and automated FL techniques. For instance, automated FL in industry faces issues such as low discrimination capacity. In addition, the obtained satisfaction results have implications for the usage and possible combination of manual, automated, and guided FL techniques.

Keywords: Controlled Experiment, Feature Location, Model-Driven Engineering, Conceptual Models

1. Introduction

Feature Location (FL) has been recognized as one of the most common activities undertaken by software developers [1]. During maintenance activities, developers need to identify where and how a feature (i.e., particular functionality) is realized in software

*Corresponding author. Tel.: +34 976060100
Email addresses: mfperez@usj.es (Francisca Pérez), jecheverria@usj.es (Jorge Echeverría), rlapena@usj.es (Raúl Lapeña), ccetina@usj.es (Carlos Cetina)

artifacts in order to fix bugs, introduce new features, and adapt or enhance existing features.

FL is considered as an important support activity during development, management, and maintenance of software, since it is helpful for a number of software tasks such as feature coverage, software reuse, program comprehension, or impact analysis. This kind of tasks are considered as a good practice by numerous major software standards such as CMMI or ISO 15504 [2], and can be critical to the success of a project [3], since they lead to increased maintainability and reliability of complex software systems [4] and decrease the expected defect rate in developed software [5].

Furthermore, as reflected in a recent survey [6], FL is gaining momentum in the research community since it helps initiate Software Product Lines (SPLs) from already existing software systems. SPLs enable a systematic reuse of variants to tailor different products. Savings of \$584 million in development costs, a 2x-4x reduction in time to market, or a reduction in maintenance costs of around 60% are among the documented real-world examples of the benefits of SPLs [7]. Hence, there is a need to adopt SPLs in companies that deal with other complex software systems such as automotive, cyber-physical and robotics [8]. To adopt a SPL, the located features are used to formalize the commonalities and variabilities across the product family. To do the formalization, feature modeling [9] can be used. In spite of the utility of FL, manual FL is a challenging activity in complex and large repositories of software artifacts that have been developed over several years by different developers [10, 2, 11]. In this context, FL activities become time-consuming and error prone [12, 13, 14, 15].

In order to reduce the effort of developers during manual FL, researchers have presented several techniques that provide automated assistance to locate features. A compendium of the most well-known techniques can be found within the survey by Julia Rubin and Marsha Chechik [16]. In the survey, the techniques are classified into static or dynamic techniques (depending on whether they involve program execution information or not) and subclassified into plain or guided techniques (depending on whether they produce an output automatically or semi-automatically with user guidance). Most of the techniques focus on FL in source code, and rely on Information Retrieval (IR) techniques to locate the features [1, 17, 18]. There are many IR techniques, but most of the research efforts show better results when applying Latent Semantic Indexing (LSI) [1, 19, 20].

Despite the importance of FL and the existence of techniques for automated assistance, research efforts tend to make comparisons between automated techniques, without comparing them against manual FL. In addition, automated techniques are focused on code, neglecting other software artifacts such as models (which have proved to increase efficiency and effectiveness in software development [21]). Thus, several important questions remain unanswered with regard to the differences in performance, productivity, and satisfaction when the manual and automated FL treatments are used to locate features in models.

To answer these questions, we conducted an experiment to compare manual FL against automated FL in models. Specifically, we recruited 18 subjects (5 experts and 13 non-experts) in the domain of our industrial partner, BSH, who has manufactured induction hobs (under the Siemens and Bosch brands, among others) for more than 15 years. For the manual FL treatment, the subjects manually located the model elements that realize a set of features using the name of each feature and models as the search space. For the automated FL treatment, we used an algorithm that leverages LSI to obtain the model elements that realize a feature description, provided by the subjects.

The experiment was conducted in terms of performance (recall, precision, and F-measure), productivity (ratio between F-measure and spent time), and satisfaction (perceived ease of use, perceived usefulness, and intention to use). Manual FL obtains average values of 44.42% recall, 42.36% precision, 41.49% F-measure, 3.43%/min productivity, 3.42 Perceived Ease of Use (PEOU), 3.47 Perceived Usefulness (PU), and 3.22 Intention to Use (ITU). Automated FL obtains average values of 76.60% recall, 14.57% precision, 22.44% F-measure, 1.21%/min productivity, 3.42 PEOU, 3.56 PU, and 3.33 ITU. After the experiment, we analyzed the results of the manual and automated FL treatments by means of a statistical analysis to find out whether significant differences exist between both. The analysis determines that the differences in performance and productivity are statistically significant, while revealing that differences in satisfaction are so minimal as to be of no practical statistical significance.

The results of our work suggest that (1) neither domain experts nor domain non-experts find the perfect solutions for the features, (2) manual FL outperforms automated FL, and (3) satisfaction results are very similar for both manual and automated FL. These issues and their causes have a number of readings and implications that can be leveraged to either improve the results for manual and automated FL (by, for instance, pairing software engineers or designing complementary artefacts for automated approaches), or to design further experiments that tackle novel research questions that arise from this work. Overall, the contributions of the paper can be summarized as follows:

- We propose an experiment for comparing manual and automated FL in models.
- We show that neither domain experts nor domain non-experts find the perfect solutions when locating features.
- We show that the use of the automated FL yields worse results than those that are manually obtained. This is a novel point that our work uncovers since research efforts have so far compared assistance tools against assistance tools in order to improve their results, instead of comparing them against humans.
- Our analysis suggests how to advance research on FL to lead to an improvement of the results for assistance tools. In addition, the findings of our work present implications for the usage and possible combination of manual, automated, and guided FL techniques.

The rest of the paper is structured as follows: Section 2 provides the necessary background in FL in models, manual FL, and automated FL. Section 3 describes the design of the experiment. Section 4 presents the results and their statistical analysis. Section 5 discusses the outcomes of our work. Section 6 deals with the threats to the validity of our work. Section 7 reviews the related work. Finally, Section 8 concludes the paper.

2. Background

2.1. Feature Location In Models

Feature Location (FL) is one of the most important and common activities performed by developers during software maintenance and evolution [22]. FL is the process of finding the set of software artifacts that realize a specific feature. FL can be performed either manually or in an automated fashion. Manual FL is a common practice but it can become error prone and time-consuming [10, 23, 8, 12, 13, 14, 15], so automated FL has received much attention during recent years [6, 16, 22] to reduce the associated manual efforts.

In addition, major players in the software engineering field foresee a broad adoption of model-driven software development techniques. Model-driven software development techniques improve the productivity and ensure the quality and performance of software in novel and industrial scenarios that demand more abstract approaches than mere coding [21]. To that extent, recent research efforts have started shifting the focus of attention towards model-based engineering [24], model-based SPL adoption [25], and feature-oriented engineering [26, 27]. These works propose approaches related to features and automated FL in the context of software engineering. However, in model-driven software development in industrial contexts, companies tend to have a myriad of products with large and complex models behind, which are created and maintained over long periods of time by different software engineers, who often lack knowledge over the entirety of the product details. Under these conditions, maintenance activities such as FL consume high amounts of time and effort, without guaranteeing good results.

For instance, take in account Figure 1. In the figure, it is possible to appreciate an excerpt of an indus-

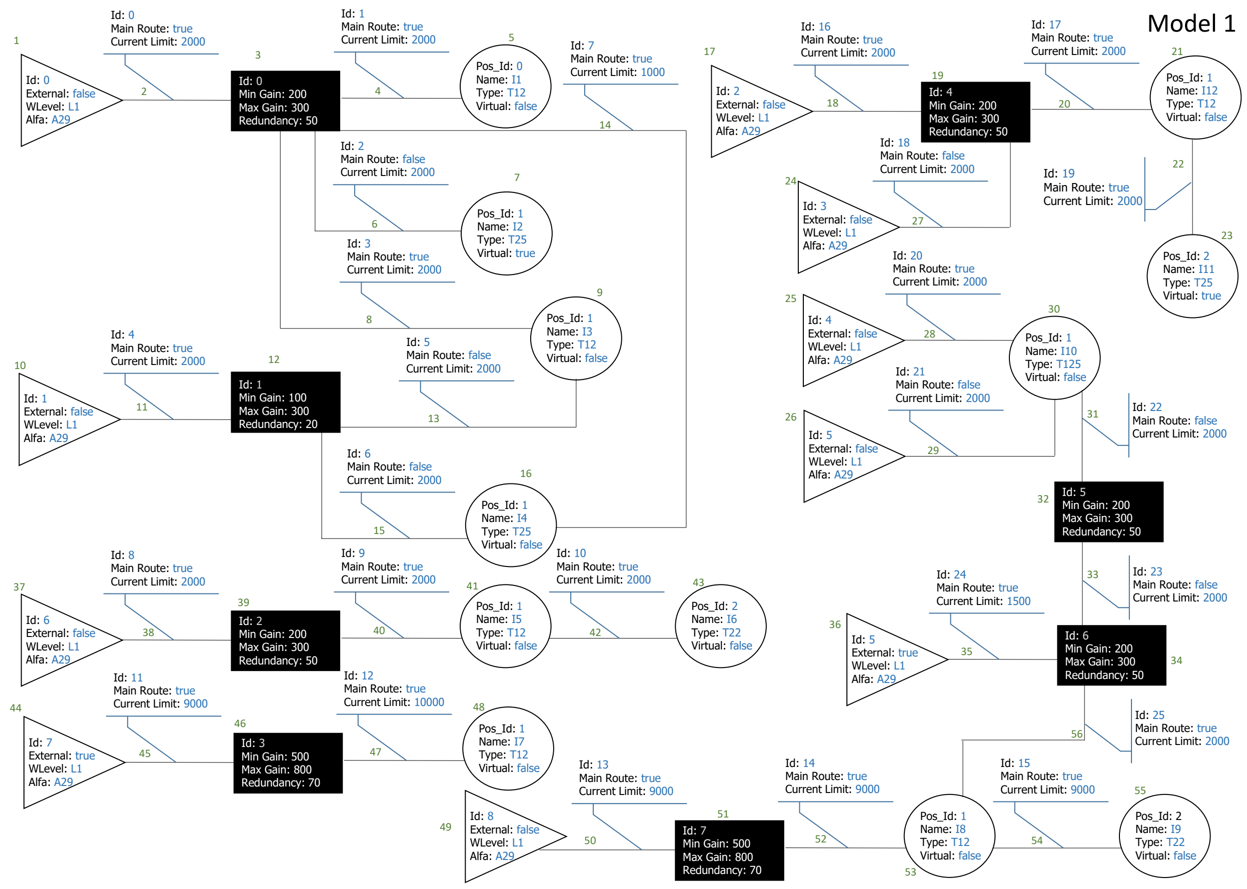

Model 1
1
Id: 0 Main Route: true Current Limit: 2000
Id: 0 External: false WLevel: L1 Alfa: A29
2
3
Id: 0 Min Gain: 200 Max Gain: 300 Redundancy: 50
Id: 1 Main Route: true Current Limit: 2000
4
5
Pos_Id: 0 Name: I1 Type: T12 Virtual: false
Id: 7 Main Route: true Current Limit: 1000
14
Id: 2 Main Route: false Current Limit: 2000
6
7
Pos_Id: 1 Name: I2 Type: T25 Virtual: true
Id: 3 Main Route: true Current Limit: 2000
8
9
Pos_Id: 1 Name: I3 Type: T12 Virtual: false
10
Id: 4 Main Route: true Current Limit: 2000
Id: 1 External: false WLevel: L1 Alfa: A29
11
12
Id: 1 Min Gain: 100 Max Gain: 300 Redundancy: 20
Id: 5 Main Route: false Current Limit: 2000
13
Id: 6 Main Route: false Current Limit: 2000
15
16
Pos_Id: 1 Name: I4 Type: T25 Virtual: false
37
Id: 8 Main Route: true Current Limit: 2000
Id: 6 External: false WLevel: L1 Alfa: A29
38
39
Id: 2 Min Gain: 200 Max Gain: 300 Redundancy: 50
Id: 9 Main Route: true Current Limit: 2000
40
41
Pos_Id: 1 Name: I5 Type: T12 Virtual: false
Id: 10 Main Route: true Current Limit: 2000
42
43
Pos_Id: 2 Name: I6 Type: T22 Virtual: false
44
Id: 11 Main Route: true Current Limit: 9000
Id: 7 External: true WLevel: L1 Alfa: A29
45
46
Id: 3 Min Gain: 500 Max Gain: 800 Redundancy: 70
Id: 12 Main Route: true Current Limit: 10000
47
48
Pos_Id: 1 Name: I7 Type: T12 Virtual: false
49
Id: 8 External: false WLevel: L1 Alfa: A29
Id: 13 Main Route: true Current Limit: 9000
50
51
Id: 7 Min Gain: 500 Max Gain: 800 Redundancy: 70
Id: 14 Main Route: true Current Limit: 9000
52
53
Pos_Id: 1 Name: I8 Type: T12 Virtual: false
Id: 15 Main Route: true Current Limit: 9000
54
55
Pos_Id: 2 Name: I9 Type: T22 Virtual: false
17
Id: 16 Main Route: true Current Limit: 2000
Id: 2 External: false WLevel: L1 Alfa: A29
18
19
Id: 4 Min Gain: 200 Max Gain: 300 Redundancy: 50
Id: 17 Main Route: true Current Limit: 2000
20
21
Pos_Id: 1 Name: I12 Type: T12 Virtual: false
Id: 18 Main Route: false Current Limit: 2000
24
Id: 3 External: false WLevel: L1 Alfa: A29
27
Id: 19 Main Route: true Current Limit: 2000
22
23
Pos_Id: 2 Name: I11 Type: T25 Virtual: true
Id: 20 Main Route: true Current Limit: 2000
25
Id: 4 External: false WLevel: L1 Alfa: A29
28
30
Pos_Id: 1 Name: I10 Type: T125 Virtual: false
Id: 21 Main Route: false Current Limit: 2000
26
Id: 5 External: false WLevel: L1 Alfa: A29
29
31
Id: 22 Main Route: false Current Limit: 2000
32
Id: 5 Min Gain: 200 Max Gain: 300 Redundancy: 50
33
Id: 23 Main Route: false Current Limit: 2000
Id: 24 Main Route: true Current Limit: 1500
36
Id: 5 External: true WLevel: L1 Alfa: A29
35
Id: 6 Min Gain: 200 Max Gain: 300 Redundancy: 50
34
56
Id: 25 Main Route: true Current Limit: 2000

Figure 1: Induction hobs industrial model excerpt

trial model, specified using a Domain Specific Language (DSL) [28]. The DSL formalizes the induction hobs manufactured by our industrial partner. The DSL has the expressiveness required to describe the functionality and regulation aspects associated to the manufactured products. Different features (i.e. particular functionalities) of the induction hobs can be located throughout the contents of the model. A feature is implemented through one or more elements of the model (i.e. a single-element or multiple-element fragment of the model). Some real-world features associated to the model in the figure are the *High External Main Power* feature, where an external inverter sends a high amount of energy through the usage of a single high current limit power channel, or the *Inductor Chain* feature, where several non-virtual inductors are linked in a series circuit fashion.

As the example might evidence, manual location of these features in the model is not a trivial task: expertise, time, and a keen eye are required to determine the appearances of the features in the model, and the fragments of the model that implement the appearances of the features. Even then, perfect feature location cannot be guaranteed, since software engineers can make mistakes (for instance, by confusing the extent of the feature and adding unnecessary elements, or by ignoring some of the properties of the elements). However, there is still a need for the location of the particular features. The desire to reduce the manual efforts and the amount of mistakes associated to feature location in these particular scenarios is what motivates the interest in automated FL techniques.

In order to better illustrate the manual and automated FL techniques, Figure 2 depicts a toy model example, specified through the same DSL used by our industrial partner, which will be used in the following sections as a running example. For the sake of understandability and in order to simplify the explanations of the two techniques, the example presents only the *High External Main Power* feature, where an external inverter sends a high amount of energy through the usage of a single high current limit power channel. In the figure, it is possible to see the name of the feature, its description, the model that contains the feature, and the set of model elements (i.e., model fragment) that implements the feature. Notice that the model fragment is highlighted in light gray in the figure and it contains two elements of the model: the inverter with ID=7, and the power channel with ID=11.

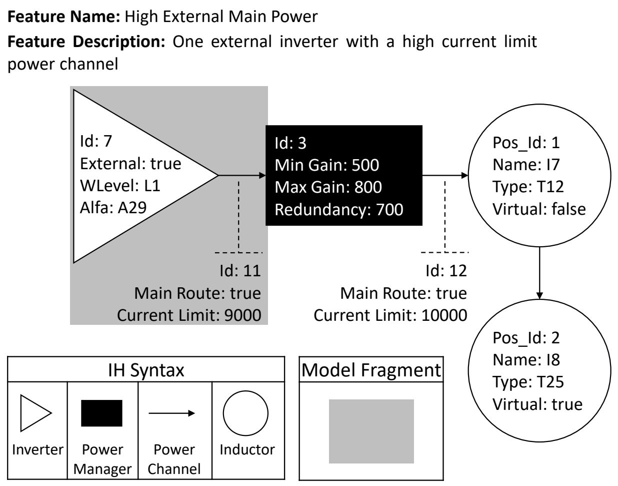


Figure 2: Example of model and model fragment

2.2. Manual Feature Location

In order to perform FL manually, a software engineer is given a feature name and a model, and has to identify the model elements in the model that are relevant for the given feature. For instance, in the running example model, the software engineer is given the model and the *High External Main Power* feature name, and has to manually identify the model elements relevant for the feature, that is, the model fragment that implements the feature. In ideal conditions, when the manual FL process is carried out perfectly by the engineer, the model fragment proposed by the software engineer will match the model fragment highlighted in light gray in the same figure. In the example, if the process is carried out perfectly by the software engineer, the result provided by the engineer will be the fragment composed by the inverter with ID=7, and the power channel with ID=11.

In order to determine whether the manual FL process is carried out without flaws for a specific feature, we utilize as the ground truth the model elements that have been added to the models throughout the years whenever the feature was added to a product. Manual FL can be leveraged by both domain experts and domain non-experts.

However, since manual FL can become error prone and time-consuming [10, 12, 13, 14, 15], automated FL has received much attention during recent years [6, 16, 22].

2.3. Automated Feature Location

In order to perform FL in an automated fashion, a software engineer is given only a feature name. The software engineer must use the feature name to produce a query, that will in turn be used as input for an automated FL technique. The style in which the query has to be written is open to the criteria of the software engineer, that is, using natural language and expressing model element names and properties using their own words. For instance, for the *High External Main Power* feature, the engineer might write down a query in the lines of 'external inverter that sends a high amount of energy through one high current limit power channel'.

As a technique for automated FL, Latent Semantic Indexing (LSI) is leveraged, since it is the technique that obtains the best results for automated FL activities [29]. The inputs in use for LSI are the provided query and one model as search space. However, the results of FL through LSI depend greatly on the style in which the natural language of software artifacts is written [30]. It is often regarded as beneficial to preprocess the inputs of LSI through natural language processing techniques. A frequent practice to achieve the preprocessing is to use a combination of parts-of-speech tagging, removal of stopwords, and stemming [31]. We adopt this practice to process the inputs. As an example, the result of processing the query provided above as an example would be 'inverter send high energy high current limit power channel'.

LSI [32] is an automatic mathematical/statistical technique that analyzes relationships between *queries* and *documents* (bodies of text). The technique has been successfully used to retrieve traceability links between different kinds of software artifacts in different contexts, specially among requirements and code [16]. The technique constructs vector representations of both a user *query* and a corpus of text *documents* by encoding them as a *term-by-document co-occurrence matrix*, and analyzes the relationships between those vectors to get a similarity ranking between the *query* and the *documents*. The outcome of LSI is used to build a model fragment from the model that serves as a candidate for realizing the query. Figure 3 shows an example *term-by-document co-occurrence matrix*, with values associated to our toy example, the vectors, and the resulting ranking. In the following paragraphs, an overview of the elements of the matrix is provided:

Each row in the matrix (*term*) stands for each of the words that compose the processed query and the natural language representation of the input model. In Figure 3, it is possible to appreciate a set of representative terms associated to our running example such as 'Inverter' or 'Inductor' as the *terms* of each row.

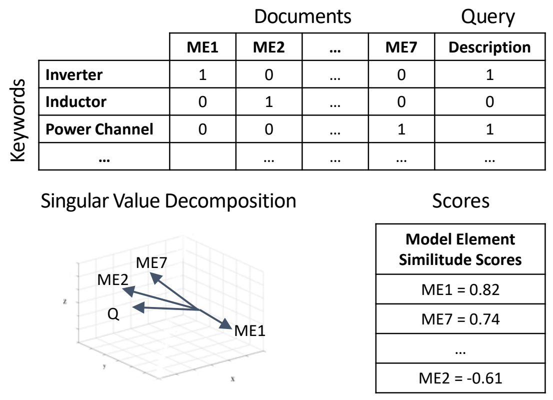
Each column (*document*) stands for one model element in the input model. In Figure 3, model elements from ME1 to ME7 stand for the *documents* associated to the elements in the running example model of Figure 2: the inverter (ME1), the two inductors (ME2 and ME3), the power manager (ME4), and the three power channels that appear in the model (ME5 to ME7). Due to space reasons, only a subset of the columns (ME1, ME2, and ME7) is represented.

The final column stands for the *query*, that is, the processed textual description of the feature derived by the software engineer from the name of the feature ('inverter send high energy high current limit power channel').

Each cell in the matrix contains the frequency with which the *term* of its row appears in the *document* denoted by its column. For instance, in Figure 3, the *term* 'Inverter' appears once in the 'ME1' *document* and once in the *query*, and the term *term* 'Inductor' appears once in the 'ME2' *document* but it does not appear in the *query* whatsoever.

Vector representations of the *documents* and the *query* are obtained by normalizing and decomposing the *term-by-document co-occurrence matrix* using *Singular Value Decomposition* (SVD) [32]. SVD is a form of factor analysis, or more properly the mathematical generalization of which factor analysis is a special case. In SVD, a rectangular matrix is decomposed into the product of three other matrices. One component matrix describes the original row entities as vectors of derived orthogonal factor values, another describes the original column entities in the same way, and the third is a diagonal matrix containing scaling values such that when the three components are matrix-multiplied, the original matrix is reconstructed.

Figure 3 depicts a three-dimensional graph of the SVD, containing the vectorial representations of some of the matrix columns. To measure the similarity degree between vectors, the cosine between the *query* vector and the *documents* vectors is calculated. Cosine values closer to one denote a higher degree of similarity, and cosine values closer to minus one denote a lower degree of similarity. Similarity increases as vectors point in the same general direction (as more *terms* are shared between *documents*). Through this measurement, the model elements are ordered ac-



	ME1	ME2	...	ME7	Description
Inverter	1	0	...	0	1
Inductor	0	1	...	0	0
Power Channel	0	0	...	1	1
...		...	...	...	...

Model Element Similitude Scores
ME1 = 0.82
ME7 = 0.74
...
ME2 = -0.61

Figure 3: Automated Feature Location Through Latent Semantic Indexing

cording to their similarity degree to the feature description, in a relevancy ranking.

The relevancy ranking (as seen in Fig. 3) is produced according to the calculated similarity degrees. In this example, LSI retrieves 'ME1' and 'ME7' in the first and second position of the relevancy ranking due to *query-documents* cosines being '0.8279' and '0.7461', implying a high similarity degree between the model elements and the feature description. On the opposite, the 'ME2' model element is returned in a latter position of the ranking due to its *query-document* cosine being '-0.6192', implying a lower similarity degree.

From the ranking, of all the model elements, only those model elements that have a similarity measure greater than a certain x value must be taken into account. A widely used heuristic is $x = 0.7$. This value corresponds to a 45° angle between the corresponding vectors. Even though the selection of the threshold is an issue under study, the chosen heuristic has yielded good results in other similar works [33, 34]. Following this principle, the elements with a similarity measure $x \geq 0.7$ are taken to conform a model fragment, candidate for realizing the query (ME1 and ME7 in the example). The model fragment generated in this manner is the final output of LSI.

3. Experiment Design

3.1. Objective

The experiment for comparing manual FL with automated FL in models was designed following the Wohlin et al. guidelines [35]. The goal of our experiment was to **analyze** FL in models, **for the purpose of** filling in the gap in empirical evaluation on this topic, **with respect to** the different FL treatments, **from the viewpoint of** both experts and non-experts in a domain, **in the context of** software development for induction hobs.

The measures used in our research to achieve the determined goal are performance, productivity, and satisfaction. To evaluate the performance we use precision, recall, and F-measure. These measures are widely accepted in the software engineering research community [36, 37]. In addition, productivity and satisfaction are widely applied when analyzing user behavior in the software engineering research community [38].

We seek to answer the following three research questions:

RQ$_1$ Is the performance different when using manual FL and automated FL for FL in software models?

RQ$_2$ Is the productivity different when using manual FL and automated FL for FL in software models?

RQ$_3$ Is the satisfaction different when using manual FL and automated FL for FL in software models?

To answer these research questions, we have formulated the following null hypotheses:

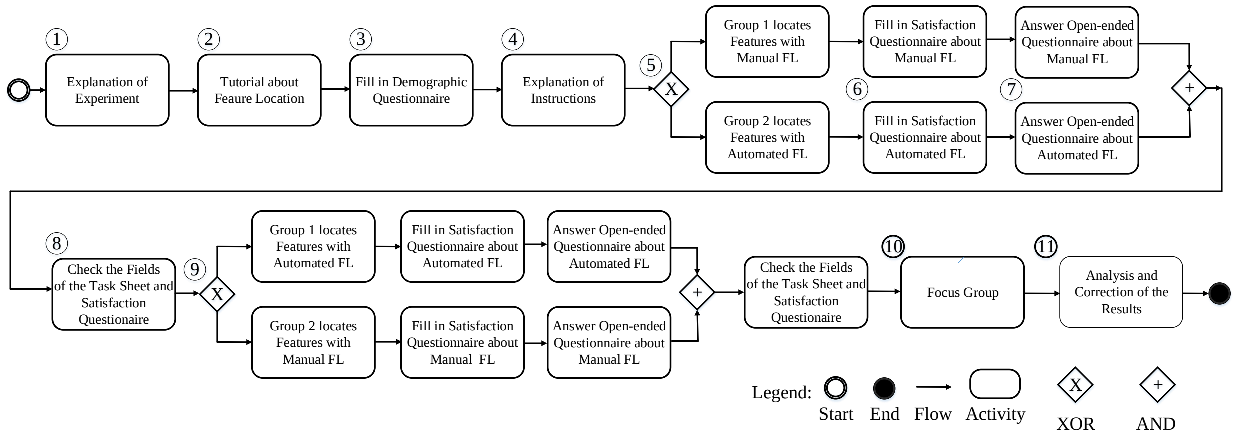
$\mathbf{H_{01}}$: There is no difference in the performance of FL in software models when manual FL and automated FL are used.

$\mathbf{H_{02}}$: There is no difference in the productivity of FL in software models when manual FL and automated FL are used.

$\mathbf{H_{03}}$: There is no difference in the satisfaction about FL in software models when manual FL and automated FL are used.

3.2. Participants

There were a total of 18 subjects, 13 of which were non-experts in the induction hobs domain, and 5 of which were experts in the induction hobs domain.



1
Explanation of Experiment
2
Tutorial about Feaure Location
3
Fill in Demographic Questionnaire
4
Explanation of Instructions
5
X
Group 1 locates Features with Manual FL
Fill in Satisfaction Questionnaire about Manual FL
Answer Open-ended Questionnaire about Manual FL
Group 2 locates Features with Automated FL
6
Fill in Satisfaction Questionnaire about Automated FL
7
Answer Open-ended Questionnaire about Automated FL
+
8
Check the Fields of the Task Sheet and Satisfaction Questionaire
9
X
Group 1 locates Features with Automated FL
Fill in Satisfaction Questionnaire about Automated FL
Answer Open-ended Questionnaire about Automated FL
Group 2 locates Features with Manual FL
Fill in Satisfaction Questionnaire about Manual FL
Answer Open-ended Questionnaire about Manual FL
+
Check the Fields of the Task Sheet and Satisfaction Questionaire
10
Focus Group
11
Analysis and Correction of the Results
Legend:
Start
End
Flow
Activity
XOR
AND

Figure 4: Overview of the experimental procedure

In order to characterize the population, the subjects filled a demographic questionnaire before entering the experiment, which allowed us to retrieve some relevant information about the working habits and expertise of the subjects of the experiment. Non-experts are master students from *Universidad San Jorge* (Zaragoza, Spain), who have spent from 0 to 13 years developing software (a mean of 4.1 years). They stated spending an average of 4.9 hours per day developing software, and 1.34 hours per day working with modeling languages. Experts are researchers and software developers in the induction hobs domain, who have spent from 3 to 13 years developing software (a mean of 6,6 years). 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.

Apart from the subjects, two instructors and one software engineer were involved in the experiment. The instructors are senior software engineers from the company who, apart from their day-to-day development activities, are responsible for training newcomer engineers after a hiring process. One instructor designed the task sheets for the experiment, provided information about the domain specific language, clarified doubts during the experiment, took notes during the focus group, generated the correction templates for the task sheets and analyzed the results. This instructor was responsible for designing the exercise so that they are similar yet independent enough to avoid the learning effect. The other instructor explained the experiment, managed the focus groups, and corrected the exercises associated to the manual FL treatment. Finally, the software engineer corrected the exercises associated to the automated FL treatment.

3.3. Defining Variables

Independent Variables. We conducted a single factor experiment where the independent variable is the treatment used to locate features in software models, which is a nominal variable with two values: manual FL and automated FL. These treatments were explained in Section 2.

In our experiment, the subjects had to perform the exercises in two different task sheets, where each task sheet contained 6 exercises regarding FL in a software model. One of the task sheets was performed through the manual FL treatment by the subjects of the experiment, and the other task sheet was performed by the automated FL treatment. The outcome of each exercise in the task sheets was a set of elements, believed by the subjects or the automated technique to be the ones that implement the feature that must be located. With manual FL, the subjects received a copy of a model along with the task sheet, and wrote down in the task sheet, for each of the 6 exercises, the elements that they consider to be the ones associated to the feature that must be located. With automated FL, the subjects wrote down a textual description of the feature to locate, and the associated elements were found with the assistance of the LSI algorithm, implemented in Java through the EJML library [39].

Dependent Variables. During the experiment, we measured performance, productivity, and satisfaction, which are defined as follows:

- **Performance** is the accuracy and completeness achieved with the two different treatments under study. We use three measurements for performance [40]:
  - *Recall* measures the number of elements according to the ground truth (the oracle) that are correctly retrieved by the proposed solution. Recall values can range from 0% (no single model element obtained from the oracle is present in the solution) to 100% (all the model elements from the oracle are present in the solution).
  - *Precision* measures the number of elements from the proposed solution that are correct according to the ground truth (the oracle). Precision values can range from 0% (no single model element from the solution is present in the oracle) to 100% (all the model elements from the solution are present in the oracle). A value of 100% precision and 100% recall implies that both the solution and the oracle are the same.
  - The *F-measure* corresponds to the harmonic mean of precision and recall.
- **Productivity** is the ratio of quality work to effort [41], we measure the quality work as the *F-measure* and the effort as the time spent (in minutes) to perform the exercises associated to each treatment. That is, we measure the *productivity* as the *F-measure* to time spent ratio (*F-measure*/time) [38].
- **Satisfaction** is measured using a satisfaction questionnaire filled out by the subjects after finishing the experiment task sheet associated with each treatment. We measure satisfaction using a 5-point Likert scale questionnaire. Based on TAM, which is a widely applied model to analyze user acceptance, the satisfaction can be decomposed as follows [40, 42]:
  - *Perceived Ease of Use (PEOU)*: the degree to which a person believes that learning and using a particular value-driven treatment would be free of effort.
  - *Perceived Usefulness (PU)*: the degree to which a person believes that using a particular treatment will increase her/his job performance within an organizational context.
  - *Intention to Use (ITU)*: the extent to which a person intends to use a particular treatment. It represents a perceptual judgment about the efficacy of the treatment, that is, whether it is cost-effective and commonly used to predict the likelihood of acceptance of a treatment in practice.

3.4. Instruments

Demographic Questionnaire. This questionnaire identifies the profile of each subject, requesting their education level, the amount of time developing software (in years), their age, their gender, the amount of time spent per day developing software, the amount of time spent per day working with software models, their knowledge about software modeling, and their knowledge about DSLs.

Software Model. The subjects had to find out the solution to the exercises in software models, or in other words, the subjects had to locate the features in the software models. Software models use the DSL that formalizes the induction hobs manufactured by our industrial partner. There are two models, which are independent from each other. The models are formed by 56 elements each, including inverters, power channels, power managers, and inductors (as shown in Figures 1 and 2).

Task sheet. A task sheet for each FL treatment was given to the subjects. Each task sheet contains 6 FL exercises and two text fields. Every exercise states a feature that must be located in a software model. Both the number of exercises in a task sheet and their difficulty are similar for both treatments. In addition, the instructor in charge of creation of the task sheets was responsible for designing the feature location exercises so that they are similar yet independent enough to avoid the learning effect. Some examples of the features that must be located as an exercise are 'quad inductor' or 'inverter power backup'. The subjects wrote down feature elements for manual FL, and a feature description for automated FL, providing us with data to calculate performance. In the text fields, the subjects had to annotate the task sheet starting and finishing times, providing us with data to calculate productivity.

Correction template. One of the instructors generated the templates with the solution for all the exercises. This instructor did not participate in the correction process. These templates were used by the other instructor and the software engineer to correct the exercises.

Satisfaction questionnaire. We defined a satisfaction questionnaire for each treatment. Even though the meaning of each question was the same for both treatments, each questionnaire includes terms that are specific to the measured treatment. The satisfaction questionnaire was built using the approach presented in [38] to evaluate PEOU, PU, and ITU. As suggested by [43], we defined six questions to measure PEOU, eight questions to measure PU, and two questions to measure ITU. The subjects had to respond to these questions using a scale of predefined scores, within the Likert scale ranging from 1 (totally disagree) to 5 (totally agree), providing us with data to calculate satisfaction.

Open-ended questionnaire. Four questions about each applied treatment were answered by the subjects. One of the questions, as an example, is: *describe your process to find out the solution.* The answers improve our knowledge about how the subjects solved the exercises of each task sheet [44].

Focus Group Interview. The objective of the focus group interview [45], composed of open questions, was to obtain qualitative data from comments of the subjects. The aims of these questions are (1) to know the rationale and the model elements more frequently used to solve the exercises, and (2) to detect the concepts or processes in the performance of the exercises that are more problematic for subjects, as well as to determinate the real causes of the problems.

The materials resulting from carrying out the experiment can be found at `https://svit.usj.es/manual-automated-fl-experiment/`.

3.5. Experimental Procedure

We chose a crossover design where the two treatments were applied. To avoid the threat of confusing treatment and order, we considered a crossover design where experimental units are divided into two groups (G1 and G2) through block randomization [38, 46]. At first, while G1 locates features in model 1 with the manual FL treatment, G2 locates features in model 1 with the automated FL treatment. Then, both groups locate features in model 2, interchanging the FL treatment. This particular experiment design uses the largest possible sample size, eliminates the learning problem effect, avoids confusing problem and treatment, and removes the variability that can appear in the results due to differences in the average responsiveness (capacity of positive and quick reaction to the experiment) among the subjects of the experiment [38].

To verify the experiment design, we conducted a pilot study [47] with one participant, who did not take part in the final experiment. We verified that the experiment had a correct parametrization, and we detected some typographical and semantic mistakes in some expressions, which were solved for the final experiment.

The experiment was conducted on two different days. On the first day, it was performed at *Universidad San Jorge* (Zaragoza, Spain) with a group of thirteen master students (non-experts) in a subject about advanced software modeling. On the second day, the same experiment set up was performed at *Universidad San Jorge* by now with five experts. In the second case, the experiment was performed by two separate groups of respectively three and two experts, based on their schedule availability. Figure 4 depicts the experimental procedure for all of the subjects, which was as follows [48]:

1. The subjects were given information about the experiment development, and were told that it was not a test of their abilities.
2. The subjects attended a tutorial (taught by the instructor) about FL in software models, and about the DSL used in the experiment. The average duration of the tutorial was 15 minutes. Copies of the slides that were used in the tutorial were given to the subjects and were available to them during the experiment.
3. The subjects were asked to fill in a demographic questionnaire prior to conducting the experiment.
4. The subjects were given a series of clear instructions to fill out the task sheet.
5. The subjects were asked to perform the exercises on the first task sheet (locating six features using one of the treatments). If the treatment was manual FL, as a result of these exercises, the subjects had to locate and write down the model elements of the features. If the treatment was automated FL, as a result of these exercises, the subjects had to write down the description of the features, and then the elements of the features were found with the assistance of an algorithm. The results were used to calculate the performance.
6. The subjects were asked to fill in a satisfaction questionnaire about the applied treatment. The answers were used to calculate the PEOU, PU, ant ITU for the applied treatment.
7. Next, the subjects answered an open-ended questionnaire. These answers improved our knowledge about how the subjects solved the task sheets.
8. When a subject finished the task sheet corresponding to one treatment, before beginning the next one, an instructor checked that the subject had filled in all of the fields in the task sheet and the satisfaction questionnaire.
9. Then the subject began the second treatment task sheet, repeating Steps 5 to 8.
10. A focus group interview about the task sheets was conducted by one instructor with each group of subjects.
11. Finally, according to the correction templates generated, the software engineer corrected the results corresponding to automated FL and one instructor corrected the results corresponding to manual FL. The other instructor analyzed the results.

4. Results

The findings of our work for each of the research questions under study can be summarized as follows:

RQ$_1$: While automated FL obtains better recall values than manual FL, manual FL outperforms automated FL overall. Differences in the results are statistically significant.

RQ$_2$: Manual FL obtains better productivity results than automated FL. Again, differences in the results are statistically significant.

RQ$_3$: Automated FL obtains generally better results than manual FL regarding satisfaction. However, differences are so minimal as to be of no statistical significance.

The following subsections provide more details on the results for each of the research questions separately.

4.1. RQ$_1$ Answer

In order to assess whether there are differences in performance when the manual and automated treatments are used to locate features in models, we measure the values of recall, precision, and F-measure. Figure 5 shows the box-and-whiskers plots for the three measurements. From left to right, up to the third plot, the figure shows: (1) the box-and-whiskers plot for *recall*, where it can be observed that the median, first quartile, and third quartile associated with automated FL are better than those associated with manual FL; (2) the box-and-whiskers plot for *precision*, where it can be observed that the median, first quartile, and third quartile associated with manual FL are better than those associated with automated FL; and (3) the box-and-whiskers plot for the *F-measure*, where it can be observed that the median, first quartile, and third quartile associated with manual FL are better than those associated with automated FL.

Table 1 shows the mean values of recall, precision, and F-measure altogether for both manual and automated FL. Regarding manual FL, the results of all the subjects yield average values of 44.42% in recall, 42.36% in precision, and 41.49% in F-measure. In the case of manual FL, domain experts achieve slightly better results than domain non-experts (60.12% in F-measure). Regarding automated FL, the results of all the subjects yield average values of 76.60% in recall, 14.57% in precision, and 22.44% in F-measure, with neither domain experts nor domain non-experts results standing out from the average.

A statistical test must be run to assess whether there is enough empirical evidence to claim that there is a difference between the two treatments regarding performance. The test should verify whether the null hypothesis $H_{\mathbf{01}}$ (as defined in Section 3) can be rejected. Statistical tests provide a probability value, p-$Value$. The p-$Value$ obtains values between 0 and 1. The lower the p-$Value$ of a test, the more likely that the null hypothesis is false. It is accepted by the research community that a p-$Value$ under 0.05 is statistically significant enough for hypothesis $H_{\mathbf{01}}$ to be considered false [49].

The statistical test that must be followed depends on the properties of the data [35]. Since our data does not follow a normal distribution in general, our analysis requires the use of non-parametric techniques. There are several tests for analyzing this kind of data, such as the Quade test or the Holm's post hoc analysis [50, 51], which are amongst the most powerful statistical tests when working with real data and a low amount of treatments. However, in order to provide an answer to the question under study (*'which of the treatments gives the best performance?'*), each treatment should be individually compared against all other alternatives [52], something which the Quade test is incapable of. Hence, we undergo a Holm's post hoc analysis, which performs a pair-wise comparison among the results of each treatment, providing a p-$Value$ that determines whether statistically significant differences exist.

With the post hoc analysis, we obtain one p-$Value$ for each of the considered measurements: 1.3×10^{-5} for recall, 4.8×10^{-6} for precision, and 9.7×10^{-5} for the F-measure. All of these values are smaller than the corresponding significance threshold value (0.05). Hence, we reject the $\mathrm{H}_{\mathbf{01}}$ hypothesis, since there are significant differences between the treatments for the performance indicators.

Once it is detected that a treatment is statistically better than another, it is also important to assess the magnitude of the improvement. *Effect size* measures are needed to analyze this. For a non-parametric effect size measure, we use Vargha and Delaney's $\hat{A}_{12}$ [53]. $\hat{A}_{12}$ measures the probability that running one treatment yields higher values than running another treatment. If the two treatments are equivalent, then $\hat{A}_{12}$ will be 0.5. For example, $\hat{A}_{12} = 0.8$ means that we would obtain better results in 80% of the cases with the first of the pair of treatments that have been compared, and $\hat{A}_{12} = 0.2$ means that we would obtain better results in 80% of the cases with the second of the pair of treatments that have been compared. In our experiment, the $\hat{A}_{12}$ values are 0.1358 for recall, 0.8611 for precision, and 0.7623 for the F-measure. The $\hat{A}_{12}$ value for recall shows a superiority of automated FL, whereas the $\hat{A}_{12}$ value for precision shows a superiority of manual FL. All in all, the $\hat{A}_{12}$ value for F-measure shows that 76.23% of the cases obtain better results with manual FL.

4.2. RQ_2 Answer

In order to assess whether there are differences in productivity when the manual and automated treatments are used to locate features in models, we measure the obtained percentage of F-measure per minute. The right-most graph of Figure 5 shows the box-and-whiskers plot for *productivity*, where it can be observed that the median, first quartile, and third quartile associated with manual FL are better than those associated with automated FL. Regarding manual FL, the results of all the subjects yield average

Table 1: Mean Values and Standard Deviations for Performance

	Manual Feature Location		
	Recall $\pm$ (σ)	Precision $\pm$ (σ)	F-measure $\pm$ (σ)
Domain non-experts	40.21 $\pm$ 23.08	33.05 $\pm$ 15.51	34.32 $\pm$ 15.72
Domain experts	55.38 $\pm$ 26.59	66.57 $\pm$ 35.52	60.12 $\pm$ 30.53
All subjects	44.42 $\pm$ 24.32	42.36 $\pm$ 26.56	41.49 $\pm$ 23.13

	Automated Feature Location		
	Recall $\pm$ (σ)	Precision $\pm$ (σ)	F-measure $\pm$ (σ)
Domain non-experts	76.27 $\pm$ 17.88	14.15 $\pm$ 10.06	22.12 $\pm$ 12.34
Domain experts	77.46 $\pm$ 23.60	14.72 $\pm$ 13.18	23.31 $\pm$ 19.03
All subjects	76.60 $\pm$ 18.89	14.57 $\pm$ 10.60	22.44 $\pm$ 13.90

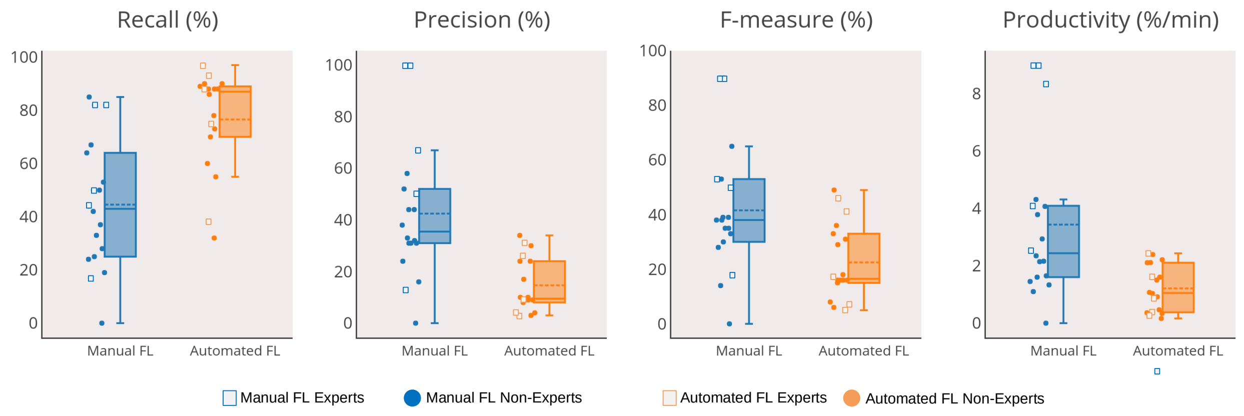


Figure 5: Box-plots for Performance (Recall, Precision, and F-measure) and Productivity

values of 3.43±2.71%/min productivity (3.54%/min for non-experts, and 3.57%/min for experts). Regarding automated FL, the results of all the subjects yield average values of 1.21±0.80%/min productivity (1.19%/min for non-experts, and 1.27%/min for experts).

We performed the same statistical analysis as in RQ_1 in order to determine whether there are statistically significant differences between manual and automated productivity. The Holm's post hoc p-$Value$ value obtained is 0.0004 (thus, differences between the techniques are significant), and an $\hat{A}_{12}$ value of 0.8117 (manual FL performs better in 81.17% of the cases). Therefore, the null hypothesis $\mathbf{H_{02}}$ can be rejected.

4.3. RQ_3 Answer

In order to assess whether there are differences in satisfaction when the manual and automated treatments are used to locate features in models, we measure the Perceived Ease of Use (PEOU), Perceived Usefulness (PU), and Intention to Use (ITU). Figure 5 shows the box-and-whiskers plots for the three measurements. From left to right, the figure shows: (1) the box-and-whiskers plot for *PEOU*, where it can be observed that the median and first quartile are higher for automated FL, whereas the third quartile is higher for manual FL; (2) the box-and-whiskers plot for *PU*, where it can be observed that the median, first quartile, and third quartile associated with automated FL are better than those associated with manual FL; and (3) the box-and-whiskers plot for *ITU*, where it can be observed that the me-

Table 2: Mean Values and Standard Deviations for Satisfaction

	Manual Feature Location		
	PEOU $\pm$ (σ)	PU $\pm$ (σ)	ITU $\pm$ (σ)
Domain non-experts	3.46 ± 0.85	3.48 ± 0.68	3.24 ± 0.92
Domain experts	3.44 ± 0.87	3.5 ± 0.68	3.24 ± 0.92
All subjects	3.42 ± 0.84	3.47 ± 0.66	3.22 ± 0.89

	Automated Feature Location		
	PEOU $\pm$ (σ)	PU $\pm$ (σ)	ITU $\pm$ (σ)
Domain non-experts	3.42 ± 0.76	3.55 ± 0.65	3.35 ± 0.90
Domain experts	3.45 ± 0.75	3.59 ± 0.65	3.35 ± 0.90
All subjects	3.42 ± 0.74	3.56 ± 0.64	3.33 ± 0.87

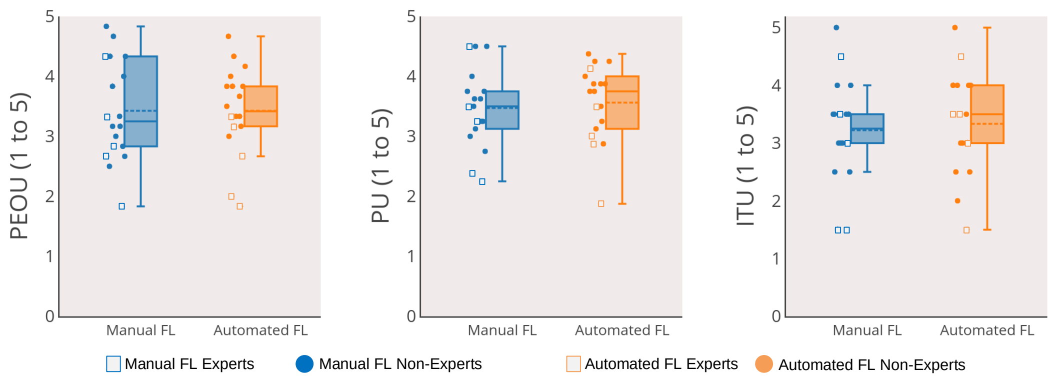


Figure 6: Box-plots for Satisfaction (Perceived Ease of Use, Perceived Usefulness, and Intention to Use)

dian, first quartile, and third quartile associated with automated FL are better than those associated with manual FL.

The top part of Table 2 shows the values associated with the three metrics for both manual and automated FL for the subjects of the experiment. Regarding manual FL, the subjects report an average *PEOU* value of 3.42, an average *PU* value of 3.47, and an average *ITU* value of 3.22. Regarding automated FL, the subjects report average values of 3.42 *PEOU*, 3.56 *PU*, and 3.33 *ITU*.

We performed the same statistical analysis to assess whether the differences between techniques are significant regarding satisfaction, and if so, by how much. In the case of satisfaction, PEOU attains a $p\text{-}Value$ of 0.98 and a $\hat{A}_{12}$ value of 0.4753, PU attains a $p\text{-}Value$ of 0.54 and a $\hat{A}_{12}$ value of 0.4414, and ITU attains a $p\text{-}Value$ of 0.84 and a $\hat{A}_{12}$ value of 0.4599. Even though automated FL obtains better results in the three measurements in slightly over 50% of the cases (according to the $\hat{A}_{12}$ values), the $p\text{-}Value$ values obtained by Holm's post hoc analysis for the three measurements point out that differences are not significant. Therefore, the null hypothesis $\mathbf{H_{03}}$ cannot be rejected.

4.4. Open-ended Questionnaire

In this subsection, we present a set of answers provided by the subjects in the open-ended questionnaires. The questionnaires were completed after finishing the task sheets with the different FL treatments (manual and automated). Some of the appearing answers are unrelated to the FL treatment in use or the characteristics of the subject. Some representative examples of those answers are: 'I am not sure about the solution', 'a greater domain knowledge would facilitate the completion of the exercises', or 'I have improved my performance with the exercises once I have already finished some of them'.

Regarding the FL process, the subjects demanded some type of filtering so that the model elements do not show all their properties. To that extent, it would be desirable for the model elements to show only the properties that are involved in the feature that is being searched for. According to this, the subjects declared that they use the properties of the elements to identify the elements that make up a feature, and as a proof of the proposed solution.

We inquired the subjects about which model element or elements they considered relevant for locating features through manual FL. Regarding non-expert subjects, the concrete syntax was considered relevant by 46% of the subjects, the element relations were considered relevant by 54% of the subjects, the element properties were considered relevant by 54% of the subjects, and the element attributes were considered relevant by 23% of the subjects. Regarding expert subjects, the concrete syntax was considered relevant by 80% of the subjects, the element relations were considered relevant by 20% of the subjects, the element properties were considered relevant by 40% of the subjects, and the element attributes were considered relevant by 20% of the subjects.

4.5. Focus Group Interview

In this subsection, the statements of the subjects in the focus group interview are summarized. The focus group interview was conducted after completing the task sheets. Most of the subjects stated that they considered manual FL to be easier than automated FL. Both experts and non-experts agreed that their performance improved as more exercises were performed. Both experts and non-experts solved the exercises by searching for the correct solutions through comparisons between all the possible solutions. Finally, expert subjects declared that the concrete syntax of the models was very important to perform the exercises.

5. Discussion

Table 3 provides an overview of the main discussion issues that arise from the results of our work. The rest of the section provides more details on each of the points outlined in the table.

Table 3: Summary of the discussion points

Experts vs Non-experts
The experts in the domain do not have complete knowledge about the features.
There is a gap between the natural language in use by the subjects and the text of the features, to which LSI is sensitive.
The textual similitude gap can be improved through a dictionary of domain translations and identification of linguistic patterns.

Manual FL vs Automated FL
None of the two treatments retrieve perfect results.
Manual FL obtains better results than automated FL in terms of performance and productivity.
Automated FL obtains better satisfaction results than manual FL.
Comparing automated FL against automated FL is not enough, it is necessary to involve manual FL to assess the FL results correctly.

The results of our work suggest that neither domain experts nor domain non-experts find the perfect solutions for the features. In addition, the performance results show indicates that while automated FL outperforms manual FL up to 32.18% in recall, manual FL outperforms automated FL up to 27.79% in precision. As recall and precision are compared against the ground truth, F-measure (the harmonic mean of precision and recall) is used. Manual FL outperforms automated FL up to 19.05% in terms of F-measure, in a statically significant manner. This is an important point that our work uncovers since research in the field has so far compared assistance tools against assistance tools but not against human participants.

Through our work, we have identified some issues inherent to manual FL and automated FL that help explain the obtained results. The following paragraphs introduce said issues and explain why automated FL fails on its duty. We also depict possible solutions to the mentioned issues, which should be explored in the future in order to improve the work possibilities of the research community dedicated to FL in models.

Regarding manual FL, both domain experts and domain non-experts seem to be unable to retrieve the appropriate solutions for the manual FL exercises. The models contain products specified and maintained for over 15 years by a team composed of several software engineers. The team has even changed over time, with newcomers taking the place of those engineers leaving the company (be it due to retirement or due to the pursue of other job opportunities). It is extremely unlikely for a single software engineer to be involved in the creation and maintenance of the full set of features and models. In addition, the long span of time for which the products under study have existed makes it hard for software engineers to remember all the details and intricacies of all the products where they have worked over time. Therefore, even domain experts tend to lack all the necessary information for the correct resolution of the manual FL exercises, having to collaborate with other experts to solve their day-to-day challenges. This issue helps explain the fact that individual software engineers are unable to find perfect solutions for the provided queries. In the light of this finding, we should study the results of pairing or grouping software engineers when performing manual FL exercises.

Regarding automated FL, utilizing a technique such as LSI yields worse results than those obtained by software engineers. Even though recall values are higher than those of software engineers, the technique obtains extremely low precision values in comparison. With automated FL, the number of model elements retrieved for a feature is very high, due to the fact that the discrimination capacity is low. On the other hand, the precision in automated FL is lower than in the case of manual FL because the amount of model elements retrieved incorrectly with automated FL is higher. This issue has passed undetected in the community so far, due to the fact that most studies compare humans against humans or tools against tools, leaving tool against human comparisons unstudied.

Upon close inspection of the queries and models, we realized why the results of the technique are not as good as the ones from human participants. LSI is based upon textual similitude, and the natural language included in the queries by the software engineers is not as technical as the language in use in the models. This is better depicted through a simple example. Consider a query built by an engineer with the text *'Choose a single virtual inverter with a high current limit power channel'*. In the model, virtual inverters are differentiated from real inverters through a Boolean value (*virtual = true*) and the current limit does not have an associated word, but rather a numeric value. Neither the Boolean value nor the numerical value appear in the query, since a human software engineer already knows how to interpret said words when looking for the solution. Therefore, the interpretation that LSI makes of the query, or in other words, what LSI tries to find for the introduced query in the model, are the terms *inverter, virtual, current limit, power channel.* Hence, LSI retrieves *all* the inverters (all of them contain the *inverter* and *virtual* terms) and *all* the power channels (all of them contain the *current limit* and *power channel* terms) in the model. Hence, it is very easy for the technique to find the elements that conform the correct solution (thus, the generally high recall values), but it is also very easy for LSI to find a lot of elements not in the correct solution (thus, the generally low precision values).

To improve the results of LSI, it would be necessary to create a dictionary of domain translations that can be used to artificially analyze the meaning of the language of the queries, reformulating them in terms of the language that appears in the models (for instance, translating *virtual* into *virtual true* to help LSI reduce the population of inverters). Another possible step towards improving automated FL would be to generate a population of model fragments from the model, defining a collection of patterns that solve specific features. Instead of building a model fragment through the analysis of individual model elements, the enhanced LSI would assess the population of model fragments, refining the resulting ranking through the patterns. Exploring these directions of automated FL and whether the obtained results help in assisting software engineers when carrying out FL exercises remains as future work.

Regarding satisfaction, the results of our work highlight a paradox: even though the results of automated FL are worse than those of manual FL, participants seem to be slightly more satisfied with automated FL outcomes than with manual FL outcomes. Participants appear to assume that tool assistance has a positive impact on the outcomes of the treatment, and they seem to not question the fact that automated FL results can be worse than what they can produce in a manual fashion. This apparent lack of skepticism and positive attitude towards automated FL might have contributed to the absence of research papers comparing manual and automated FL so far.

However, while most of the literature puts the focus on FL in source code, our work is centered in reporting FL in models. Additional experimentation is needed to determine whether the results of studying FL in models can be transferred, generalized, or compared against the results from other works in the literature that focus on FL in source code.

In any case, models are becoming increasingly popular [21], and a widespread and extensively utilized software artifact in novel approaches related to model-based engineering [24], model-based SPL adoption [25], and feature-oriented engineering [26, 27]. These works show that FL in models remains as a relevant topic for the research community. In that sense, the findings of our work contribute to the study of the application of manual and automated FL in models to the context of FL techniques and their application in models. Other researchers in the context can use the provided findings to perform more experimentation in the field, as well as to improve the FL techniques and their usage.

Finally, although our work puts the focus on manual and automated FL techniques, guided FL techniques have been gaining traction in the last years. These semi-automatic techniques, which can be found in the survey by Julia Rubin and Marsha Chechik [16], generally work by producing a set of results and then allowing the software engineers to iteratively re-adjust the query, the result set, or both. However, our work suggests that the subjects are equally satisfied with the results of manual and automated FL techniques. This finding suggests that, when faced with the results provided by an automated technique, the subjects consider them to be at least as good as the final version of the results that they can produce by themselves. 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, beneficial as they are.

This issue, in turn, raises several research questions. What is the predisposition of the subjects to acknowledging their own mistakes? Do subjects recognize their own limitations when performing manual FL? Are performance, productivity, and satisfaction of subjects improved when using guided FL techniques to perform the FL activity in models? And more importantly, when do subjects consider results to be good enough so as to disregard iteration over queries and results? It becomes clear that more experimentation is needed to explore this point and the implications of the results of our work with regards on how to combine manual, automated, and guided FL efforts. Perhaps the results of manual FL techniques could be used as seeds that can be leveraged as inputs for automated techniques, developing novel ways of approaching guided FL. Nonetheless, responding to these uncovered research questions, issues, and their implications remains as an open topic. Designing and carrying out the necessary experimentation constitutes the next steps in our future work.

6. Threats To Validity

To describe the threats of validity of our work, we use the classification of [54], which distinguishes four aspects of validity (construct validity, internal validity, external validity, and reliability).

Construct validity reflects the extent to which the operational measures that are studied represent what the researchers have in mind and what is investigated based on the research questions. There are six threats of this kind: author bias, task design, mono-method bias, hypothesis guessing, evaluation apprehension, and mono-operation bias. *Author bias* occurs when people that define the artifacts can subjectively influence the results that they are looking for. In order to mitigate this threat, the models and features to locate are balanced, i.e. their sizes are the same for both location treatments. The *task design* threat appears when the exercises in the task sheets can be correctly performed just by chance. To mitigate this threat, the proposed exercises did not have a true/false answer. Rather, the subjects had to locate a feature, which is very difficult for subjects to answer correctly if they do not understand the exercise. *Mono-method bias* occurs due to using a single type of measure [38]. All of the measurements were affected by this threat. To mitigate this threat for the performance and productivity measurements, an instructor checked that the subjects filled all the fields correctly and we mechanized these measurements as much as possible by means of correction templates. Regarding satisfaction, we mitigated this threat by using a widely applied model (TAM) [40, 42]. The *hypothesis guessing* threat appears when the subject may guess the hypotheses and work having them in mind. To mitigate this, we did not speak with the subjects about the research questions or the objective of the experiment. The *evaluation apprehension* threat appears when the subjects are afraid of being evaluated. To mitigate this threat the subjects were informed that the experiment was not a test of their abilities. Finally, *mono-operation bias* occurs when treatments depend on a single operationalization. The experiment was affected by this threat, since we worked with a DSL. For this reason, generalization of results must be approached with caution.

Internal validity threats appear when causal relations are examined, since there is a risk that the studied aspects may be affected by other factors that are not considered in the experiment. There are seven threats of this kind: learning effect, information exchange, understandability, fatigue effects, researcher bias, imbalanced group of subjects and subject motivation. A *learning effect* occurs when the subjects learn something during the experiment that may influence later exercises. We mitigated this threat by ensuring every participant worked with the two FL treatments on two different experimental objects. Regarding the *information exchange* threat, since the experiment was designed to take place in two different days, the subjects might have been able to exchange information during the time between the sessions. To our knowledge, experts and non-experts do not know each other. The *understandability* threat appears when the subjects do not understand how to carry out the experiment. This threat was mitigated by writing the experimental materials in the language of subjects or easy English. In addition, a tutorial was taught before the experiments by instructors who were also available to solve doubts. The *fatigue effect* occurs when the subjects get tired during the experiment. This was solved by establishing a total time of 90 minutes for the whole experiment (including the tutorial). *Researcher bias* occurs when assumptions and personal beliefs of the researcher might affect the study [55]. We mitigated this threat by not participating in the selection of the participants; on the other hand, to avoid the influence in the correction of the exercises, one instructor designed the correction templates and he didn't participate in the correction process. Our experiment was affected by the threat of an *imbalanced group of subjects* because the amount of experts was lower than the amount of non-experts. We mitigated this threat by using block randomization, in which randomization among treatments (manual and automated FL) was performed separately for non-experts and experts [46]. The main reason for the imbalance in the groups of subjects (that is, having less subjects in the experts group than in the non-experts group) is the difficulty to access the experts. Even though the number of experts may seem relatively small, the user experience research advises to use five subjects in the usability test to detect 80% of the usability problems [56]. Finally, *subject motivation* threats appear when the subjects are not motivated to participate in the experiment. The experiment was affected by this threat, since non-experts carried it out as a non-optional activity during a masters degree subject. To mitigate this threat, the score of non-experts in the subject was augmented by their participation in the experiment.

External validity threats are concerned with to what extent it is possible to generalize the findings and to what extent the findings are of relevance for other cases. There are four threats of this kind: statistical power, object dependency, and selection subjects. The *statistical power* threat appears when the number of subjects is not enough to generalize results. Our experiment was affected by this threat, because the number of subjects (18) was not high enough to generalize results. To mitigate this threat, we have used a confidence interval where conclusions are 95% representative. The *object dependency* threat appears when the results may depend on the objects used in the experiment, thus not being generalizable. The experiment was affected by this threat since we only analyzed exercises with models belonging to the induction hobs domain. The *influence of the domain* threat appears when the outcomes depend on a specific domain. This experiment was affected by this threat since we only analyzed the induction hobs domain. Finally, regarding the *selection subjects* threat, the experiment was performed by non-experts and experts. The participation of non-experts can be a source of experiment weakness. However, using students as subjects instead of domain experts is not a major issue as long as research questions are not specifically focused on experts [57].

Reliability is concerned with to what extent the data and the analysis are dependent on the researchers. There are two threats of this type: data collection, and completion data. The *data collection* threat appears when data collection is not carried out in the same way throughout the different sessions. This was mitigated by applying the same procedure to each session and using the same formulas to calculate the dependent variable values. The *completion data* threat appears when there is missing data after the data collection process. To mitigate this threat, two instructors tested the data coherence when the subjects had finished the exercises in each task sheet.

In this experiment, as far as possible, the threats have been avoided or mitigated. Even though the experiment uses real-world data from an industrial partner, the number of subjects is not enough to guarantee the generalization of the results, and thus they should be extrapolated with caution. However, we consider replicating the experiment on a large scale with more experienced and professional developers. For instance, the experiment can be conducted again with additional non-experts coming from future editions of the master's degree, and with more experts coming from our industrial partner. The replications of the experiment should consider the dimensions (Operationalization, Population, Protocol and Experimenters) that are identified by Gómez et al. [58]. To replicate the experiment on a large scale with more experienced and professional developers, only the dimension Population should be changed with an increased number of subjects, who will be classified as either expert or non-expert. We also consider replicating the experiment using other DSLs coming from other industrial domains and partners. To do this, only the dimension Protocol should be changed to vary the software models and the task sheets, whereas their size and difficulty should be the same than in our experiment.

7. Related Work

Some works research how developers locate features. The work presented in [59] reports an exploratory study of FL, consisting of three experiments with six FL exercises. The study evaluates the quality of FL and the impact of explicit FL knowledge, also proposing a conceptual framework for understanding FL processes. In [60], the authors present an exploratory case study on identifying and manually locating features in Marlin, a variant-rich open-source embedded firmware. Another work [20] presents a novel feature location technique named SITIR, a semi-automated technique for FL in source code. These empirical studies do not compare manual FL versus automated FL as our work does.

Other works compare different FL treatments. In [61], a case study is described where two features must be located in a sample of poorly structured legacy Fortran code through three different FL treatments: software reconnaissance, dependency graphs, and "grep" text search, finding out advantages and disadvantages for every treatment. In [62], the authors introduce rank topology, a metric for comparing FL techniques, and propose an alternative measure of relevance based on the likelihood of finding results in a ranking. In [63], the authors compare five different Single Objective Evolutionary Algorithms for FL in models. The work presented in [64] takes advantage of long-living software systems to address the FL challenge, using commonality and modifications through model retrospectives to promote model fragments that suffered less modifications over time. Another paper [1] proposes a new FL technique named PROMESIR, combining

LSI over source code with a Scenario-based Probabilistic Ranking of events. The approach is validated with the Eclipse and Mozilla projects, measuring effectiveness and comparing PROMESIR with other two existent techniques. These works do not perform a formal evaluation with users as we do.

Other research focuses on improving FL. Ji et. al. [65] argue that manual recording and maintaining of features can be effectively embedded into software assets, and that costs are surpassed by the benefits of the information during development. They test this hypothesis in a study where they simulate the development of a product line using a lightweight code annotation approach, identifying annotation evolution patterns and measuring annotation cost against benefit. In [66], an exploratory study is performed with Motorola to compare the CodeTEST and Klocwork inSight tools, identifying the pitfalls that might make this tool combination unusable for FL. The work presented in [67] focuses on FL in a complex large-scale mature software system, developed by professional software engineers. In [68], a tool named FLOrIDA is presented and evaluated by two experts. The tool uses previously defined annotations to locate and visualize features in a large software industrial system. Again, these works lack a formal evaluation with users.

Finally, as asserted in the background, other recent research efforts have started shifting the focus of attention towards model-based engineering [24], model-based SPL adoption [25], and feature-oriented engineering [26, 27]. These works propose approaches related to features and automated FL in the context of software engineering. Our work does not propose novel automated FL approaches, but rather puts the focus on comparing manual FL approaches against automated FL approaches such as the ones presented by these works in the context of industrial software engineering. To our knowledge, there is no previous research that compares manual and automated FL in models.

8. Conclusions

Feature Location (FL) is one of the most frequent activities in software development, particularly during maintenance activities. However, in industrial environments, software artifacts are developed over long periods of time by different software engineers, resulting in complex and large repositories, and thus FL becomes a challenging, time-consuming activity that does not guarantee good results. To tackle this issue, researches have proposed automated Information Retrieval techniques such as LSI to assist software engineers in FL.

However, these automated FL techniques are often compared against other techniques, neglecting comparisons against the manual FL process. Moreover, most of these techniques tackle code, ignoring other relevant artifacts such as models. In this paper, we have compared manual FL against automated FL in models through an experiment that compares the results obtained by human subjects against those obtained by the usage of LSI. The experiment was conducted in terms of performance, productivity, and satisfaction. While manual FL obtains better results in terms of performance and productivity, subjects are slightly more satisfied with automated FL usage and results. The results of our work suggest that (1) neither domain experts nor domain non-experts find the perfect solutions for the features, (2) manual FL outperforms automated FL, and (3) satisfaction results are very similar for both manual and automated FL.

The analysis of these results suggests discussion points, findings and future directions to lead an improvement of FL techniques (manual, automated, and guided FL). For example, the results can be better than those manually obtained by exploring the direction of the discrimination capacity of existing FL techniques such as LSI. Other directions to be explored are pairing software engineers or designing complementary artefacts for automated approaches. Following these directions can help industry to reduce the time that is necessary to locate features in complex and large repositories of software artifacts that have been developed over several years by different developers. Thus, the adoption of automated FL techniques can be promoted in industry. Furthermore, the future directions from this work lead the design of further experiments that tackle novel research questions.

Acknowledgements

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] 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 (2007) 420–432.

[2] R. Oliveto, M. Gethers, D. Poshyvanyk, A. De Lucia, On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery, in: 2010 IEEE 18th International Conference on Program Comprehension, IEEE, 2010, pp. 68–71.

[3] R. Watkins, M. Neal, Why and How of Requirements Tracing, IEEE Software 11 (1994) 104–106.

[4] A. Ghazarian, A Research Agenda for Software Reliability, IEEE Reliability Society 2009 Annual Technology Report (2010).

[5] P. Rempel, P. Mäder, Preventing Defects: the Impact of Requirements Traceability Completeness on Software Quality, IEEE Transactions on Software Engineering 43 (2017) 777–797.

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

[7] Key Benefits: Why Product Line Engineering?, `http://www.productlineengineering.com/benefits/key-benefits.html`, 2020.

[8] T. Berger, J.-P. Steghöfer, T. Ziadi, J. Robin, J. Martinez, The State of Adoption and the Challenges of Systematic Variability Management in Industry, Empirical Software Engineering (2019).

[9] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report, Carnegie-Mellon University Software Engineering Institute, 1990.

[10] J. Krüger, T. Berger, T. Leich, Features and how to find them: A survey of manual feature location, in: Software Engineering for Variability Intensive Systems - Foundations and Applications, Taylor & Francis Group, 2019, pp. 153–172.

[11] Y. Zhang, R. Witte, J. Rilling, V. Haarslev, Ontological Approach for the Semantic Recovery of Traceability Links Between Software Artefacts, IET software 2 (2008) 185–203.

[12] C., L. O'Brien, Map - mining architectures for product line evaluations, in: Proceedings Working IEEE/IFIP Conference on Software Architecture, 2001, pp. 35–44.

[13] P. Grünbacher, R. Rabiser, D. Dhungana, M. Lehofer, Model-based customization and deployment of eclipse-based tools: Industrial experiences, in: 2009 IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 247–256.

[14] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, Advancing candidate link generation for requirements tracing: the study of methods, IEEE Transactions on Software Engineering 32 (2006) 4–19.

[15] 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 Trans. Softw. Eng. 32 (2006) 971–987.

[16] J. Rubin, M. Chechik, A Survey of Feature Location Techniques, in: Domain Engineering, Springer, 2013, pp. 29–58.

[17] A. Marcus, J. I. Maletic, Recovering documentation-to-source-code traceability links using latent semantic indexing, in: Proceedings of the 25th International Conference on Software Engineering, ICSE '03, 2003, pp. 125–135.

[18] W. Zhao, L. Zhang, Y. Liu, J. Sun, F. Yang, Sniafl: towards a static non-interactive approach to feature location, in: Proceedings. 26th International Conference on Software Engineering, 2004, pp. 293–303.

[19] M. Revelle, B. Dit, D. Poshyvanyk, Using data fusion and web mining to support feature location in software, in: IEEE 18th International Conference on Program Comprehension (ICPC), 2010, pp. 14–23.

[20] D. Liu, A. Marcus, D. Poshyvanyk, V. Rajlich, Feature location via information retrieval based filtering of a single scenario execution trace, in: Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, ASE '07, ACM, 2007, pp. 234–243.

[21] M. Brambilla, J. Cabot, M. Wimmer, Model-driven software engineering in practice, Synthesis Lectures on Software Engineering 1 (2012) 1–182.

[22] B. Dit, M. Revelle, M. Gethers, D. Poshyvanyk, Feature Location in Source Code: a Taxonomy and Survey, Journal of software: Evolution and Process 25 (2013) 53–95.

[23] Extractive Software Product Line Adoption Catalog, `https://but4reuse.github.io/espla_catalog/ESPLACatalog.html`, 2020.

[24] 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, ACM, 2016, pp. 272–282.

[25] 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.

[26] J. Martinez, T. Ziadi, M. Papadakis, T. F. Bissyandé, J. Klein, Y. Le Traon, Feature location benchmark for extractive software product line adoption research using realistic and synthetic eclipse variants, Information and Software Technology 104 (2018) 46–59.

[27] J. Krüger, M. Mukelabai, W. Gu, H. Shen, R. Hebig, T. Berger, Where is my feature and what is it about? a case study on recovering feature facets, Journal of Systems and Software 152 (2019) 239–253.

[28] A. van Deursen, P. Klint, J. Visser, Domain-specific languages: An annotated bibliography, SIGPLAN Not. 35 (2000) 26–36.

[29] S. Winkler, J. Pilgrim, A Survey of Traceability in Requirements Engineering and Model-Driven Development, Software and Systems Modeling (SoSyM) 9 (2010) 529–565.

[30] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, T. Menzies, Automatic Query Reformulations for Text Retrieval in Software Engineering, in: Software Engineering (ICSE), 2013 35th International Conference on, IEEE, 2013, pp. 842–851.

[31] A. Hulth, Improved Automatic Keyword Extraction Given More Linguistic Knowledge, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2003.

[32] T. K. Landauer, P. W. Foltz, D. Laham, An Introduction to Latent Semantic Analysis, Discourse Processes 25 (1998) 259–284.

[33] A. Marcus, A. Sergeyev, V. Rajlich, J. I. Maletic, An Information Retrieval Approach to Concept Location in Source Code, in: Proceedings of the

11th Working Conference on Reverse Engineering, 2004, pp. 214–223.

[34] H. E. Salman, A. Seriai, C. Dony, Feature Location in a Collection of Product Variants: Combining Information Retrieval and Hierarchical Clustering, in: The 26th International Conference on Software Engineering and Knowledge Engineering, 2014, pp. 426–430.

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

[36] G. Salton, M. J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, Inc., 1986.

[37] J. Echeverria, F. Pérez, C. Cetina, O. Pastor, Assessing the performance of automated model extraction rules, in: Information Systems Development: Advances in Methods, Tools and Management (ISD2017 Proceedings), 2017.

[38] J. I. Panach, S. España, O. Dieste, O. Pastor, N. Juristo, In search of evidence for model-driven development claims, Inf. Soft. Techn. 62 (2015) 164–186.

[39] P. Abeles, EJML, `ejml.org`, 2019.

[40] E. Souza, S. Abrahao, A. Moreira, E. Insfram, J. Araujo, Evaluating the efficacy of value-driven methods: a controlled experiment, in: Information Systems Development: Advances in Methods, Tools and Management (ISD 2017 Proceedings) Larnaca, Cyprus: University of Central Lancashire Cyprus., 2017.

[41] IEEE, ISO/IEC/IEEE international standard - systems and software engineering – vocabulary, ISO/IEC/IEEE 24765:2010(E) (2010) 1–418.

[42] F. D. Davis, Perceived usefulness, perceived ease of use, and user acceptance of information technology, MIS Q. 13 (1989) 319–340.

[43] D. L. Moody, The method evaluation model: a theoretical model for validating information systems design methods, in: ECIS, 2003.

[44] T. Lethbridge, S. E. Sim, J. Singer, Studying software engineers: Data collection techniques for software field studies, Empirical Software Engineering 10 (2005) 311–341.

[45] R. A. Krueger, M. A. Casey, Focus groups: A practical guide for applied research, Sage publications, 2014.

[46] H. J. Seltman, Experimental design and analysis, Online at: http://www.stat.cmu.edu/ hseltman/309/Book/Book.pdf (2012).

[47] M. Asadi, S. Soltani, D. Gašević, M. Hatala, The effects of visualization and interaction techniques on feature model configuration, Empirical Software Engineering (2014) 1–38.

[48] J. Echeverría, F. Pérez, J. I. Panach, C. Cetina, O. Pastor, The influence of requirements in software model development in an industrial environment, in: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017, pp. 277–286.

[49] A. Arcuri, L. Briand, A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering, Softw. Test. Verif. Reliab. 24 (2014).

[50] W. Conover, Practical nonparametric statistics., 3rd ed., John Wiley & Sons, 1999.

[51] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power, Information Sciences 180 (2010) 2044–2064.

[52] L. Arcega, J. Font, Ø. Haugen, C. Cetina, An approach for bug localization in models using two levels: model and metamodel, Software & Systems Modeling (2019) 1–26.

[53] A. Vargha, H. D. Delaney, A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong, Journal of Educational and Behavioral Statistics 25 (2000).

[54] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14 (2009) 131–164.

[55] R. Feldt, A. Magazinius, Validity threats in empirical software engineering research-an initial survey., in: SEKE, 2010, pp. 374–379.

[56] J. Nielsen, Why you only need to test with 5 users, 2000.

[57] 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 (2002) 721–734.

[58] O. S. Gómez, N. Juristo, S. Vegas, Understanding replication of experiments in software engineering: A classification, Information and Software Technology 56 (2014) 1033 – 1048.

[59] W. Jinshui, P. Xin, X. Zhenchang, Z. Wenyun, How developers perform feature location tasks: a human-centric and process-oriented exploratory study, Journal of Software: Evolution and Process 25 (2013) 1193–1224.

[60] 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, VAMOS 2018, ACM, New York, NY, USA, 2018, pp. 105–112.

[61] 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 (2003) 105 – 114.

[62] E. Hill, A. Bacchelli, D. W. Binkley, B. Dit, D. J. Lawrie, R. Oliveto, Which feature location technique is better?, in: 2013 IEEE International Conference on Software Maintenance, 2013, pp. 408–411.

[63] 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 (2017).

[64] 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.

[65] 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, SPLC '15, ACM, New York, NY, USA, 2015, pp. 61–70.

[66] S. Simmons, D. Edwards, N. Wilde, J. Homan, M. Groble, Industrial tools for the feature location problem: an exploratory study, Journal of Software Maintenance 18 (2006) 457–474.

[67] H. Jordan, J. Rosik, S. Herold, G. Botterweck, J. Buckley, Manually locating features in industrial source code: The search actions of software nomads, in: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, ICPC '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 174–177.

[68] B. Andam, A. Burger, T. Berger, M. R. V. Chaudron, Florida: Feature location dashboard for extracting and visualizing feature traces, in: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems, VAMOS '17, 2017, pp. 100–107.