Impact of Footprinting on Model Quality: An Experimental Evaluation

Cédric Jeanneret and Martin Glinz
University of Zurich
Zurich, Switzerland
{*jeanneret, glinz*}*@ifi.uzh.ch*

Benoit Baudry and Benoit Combemale
IRISA
Rennes, France
{*benoit.baudry, benoit.combemale*}*@irisa.fr*

***Abstract*—When modeling requirements, software analysts have to choose the relevant modeling constructs among all those available. If they do not choose the right set, their model may lack some important information or their model may contain many superfluous details. In previous work, we proposed to capture the purpose of a model with a set of model operations such as queries or model transformations. Then, modelers can analyze the footprints of these operations, that is, the set of model elements touched during their execution.**

In this paper, we report on two controlled experiments performed with students to evaluate whether footprinting can help them in creating better models. While our studies did not demonstrate statistically significant benefits of footprinting, they reveal the importance of training and tool support for the analysis of footprints.

***Keywords*-requirements modeling, model footprinting, controlled experiment**

I. Introduction

With the advent of Model Driven Engineering (MDE), models play an important role in the development of software. Conceptually, a model is an abstract representation of an original for a given purpose. Ideally, a model exactly represents its *domain*, that is, the set of all possible statements that would be correct about the original and relevant for the purpose at hand [1].

A model can differ from its domain in two different ways. In [1], Lindland et al. define validity as the extent to which a model only contains statements from the domain. A model with superfluous details may be harder to understand and is probably more expensive to create than necessary. On the opposite, a model may lack some relevant details, making it incomplete. When using an incomplete model, an interpreter may draw wrong conclusions about the original.

Thus, the value of a model depends heavily on whether it contains all relevant information and only this information. However, creating models that actually represent their domain is not an easy task. Eliciting information about the original is not enough; the modeler needs to understand the purpose of the model as well. Often, little is known about this purpose and typical modeling assignments only mention the modeling language to be used. When the language contains many constructs, as UML does, this indication offers little help.

Recently, we have invented a technique called footprinting to detect the presence of superfluous elements in models [2]. In a MDE setting, the purpose of a model can be characterized by the set of model operations (e.g., queries, view extractions or model transformations) that the model must enable. A *footprint* is the set of all model elements that have been used during the execution of these operations. When footprints are highlighted, modelers can easily find the elements that were not used by the set of operations executed on the model, which gives hints about their possible irrelevance. However, the benefits of footprinting have not yet been evaluated empirically.

In this paper, we are interested in the actual effect of footprinting on the quality of models. To investigate this effect, we performed a pair of controlled experiments: one at the University of Zurich, Switzerland, involving undergraduate students enrolled in a basic Software Engineering course, and one at the University of Rennes 1, France, involving graduate students enrolled in a course on MDE techniques.

The remainder of the paper is structured as follows. In the next section, we provide background information about model quality and model footprinting. We describe our experiments in Section III, while their results are presented and analyzed in Section IV. Finally, in Section V, we discuss our findings and we conclude in Section VI.

II. Background

A. Model Quality

As models become more and more important for the development of software, many researchers proposed frameworks for evaluating and improving the quality of models. In [1], Lindland et al. proposed the first systematic approach to identifying quality goals and means to achieve them. In this framework based on semiotics (the study of symbols), models are seen as sets of statements. Models are compared to three other sets of statements: The *language* is the set of statements expressible in the modeling language. The *domain* is the set of all possible statements that would be correct and relevant while the *audience interpretation* is the set of statements that the model users think a model contains.

Their framework defines three types of model quality. *Syntactic quality* is how well a model corresponds to the

language. *Semantic quality* is how well the model corresponds to the domain. Finally, *pragmatic quality* is how well a model corresponds to the audience interpretation. We chose the framework of Lindland et al. for our work because it laid the foundation for many other publications. Krogstie et al. extended this framework with additional constructs from the semiotics theory [3]. Moody et al. validated the framework empirically in [4] while España et al. used the framework to compare two RE methods in [5].

Lindland et al. distinguish two semantic quality goals: completeness and validity. A model is *complete* if it contains all the statements from the domain, while a model is *valid* if it only contains statements from the domain (and nothing else). A statement can be invalid for two reasons: it is incorrect or it is irrelevant for the purpose at hand. In our work, we distinguish between the two cases. We use the term *confinement* for the extent to which a model only contains relevant statements [6] while *correctness* designates the extent to which a model only contains correct information.

In typical settings, the domain does not exist explicitly. Semantic quality is then typically assessed subjectively with reviews. In our experiments, the domain is represented by a reference model. This has been done previously by España et al. to compare two RE methods in [5]. Furthermore, we not only consider the actual quality of models — measured on the model with respect to the reference model — but also the quality as it is perceived by the modelers. This perception is important, because on a typical modeling task, the domain does not exist explicitly and there is no such thing as a reference model. Thus, modelers consider their modeling task complete when their models contain all statements from what they perceive as the domain.

B. Model Footprinting

To assess the relevance of some model elements, the modelers must know the purpose of their models. In previous work [7], we proposed to use goal oriented requirements engineering methods to capture this purpose. Eventually, this purpose must be operationalized with some model operations such as analyses, queries and model transformations. For example, a class diagram could be used to derive a glossary of a domain or a state machine could be used to generate the skeleton of an implementation. The main assumption behind the idea of footprinting is that the purpose of a model can be characterized by the set of model operations that will be performed on this model.

These operations can be performed mentally by humans or executed by a computer. In both cases, a model must conform to some structure to support these operations. This structure is typically expressed with an object-oriented metamodel. Such a metamodel defines the types and features — meta-classes, meta-attributes and meta-references — that can be used in a model. For example, the metamodel of UML defines meta-classes such as `Class` and `State`, and meta-attributes such as `name` and `isAbstract`.

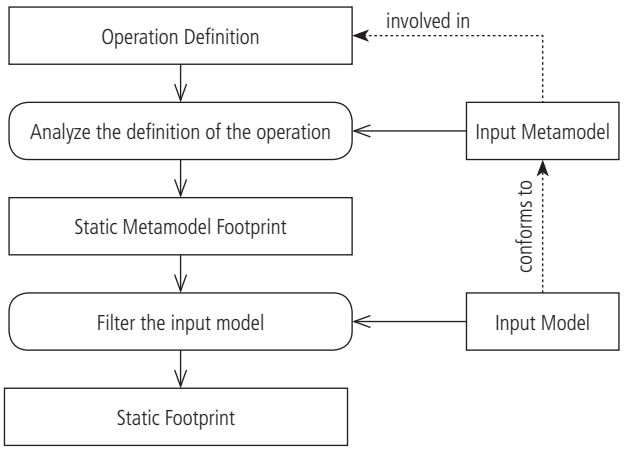


Figure 1: Static Footprinting

Footprinting consists of finding which elements of a model are used by a given model operation during its execution. There are two techniques for footprinting: dynamic and static footprinting [2]. Dynamic footprinting analyzes the execution of a set of operations, while static footprinting estimates the footprint based on a static analysis of the definition of the operations. In [2], we demonstrated that, despite some limitations, static footprints are very precise estimates of dynamic footprints and yet cheap to compute.

In the experiments, we have used static footprinting, as it can be done without tool support. Static footprinting is divided into two phases (see Figure 1). First, the source code of the operations is analyzed. Types and features are collected along their control flow graph. This analysis yields a *static metamodel footprint*, the set of all constructs relevant for the operations being analyzed. In the second phase, models are filtered through the static metamodel footprint: only elements related to constructs in the static metamodel footprint are kept in the *static (model) footprint*. This filtering is very simple and can be done manually. Thus, as long as the static metamodel footprint is provided, static footprinting can be done without tool support.

For example, the operation generating a glossary from a class diagram may be implemented as a model-to-text transformation. The source code of this operation may involve classes and generalizations, but probably not ports or dependencies. In addition, the operation will likely read the name of a class, but not whether it is abstract. The static metamodel footprint would then include the following elements:

- `Class`
  - `name`
- `Generalization`

The modelers should only model elements that are instances of the static metamodel footprint. Any other elements will not be used by the operations performed on the model. They may therefore be irrelevant, decreasing the confinement of the model. Furthermore, the modeler should consider all constructs in the static metamodel footprint. If a relevant meta-class or meta-attribute is omitted from the model, some information is probably missing, decreasing the completeness of the model. The filtering step of static footprinting validates whether a model only contains elements related to the static metamodel footprint.

We believe that with the help of footprinting, modelers produce models that are more confined and more complete than without footprinting. Indeed, static metamodel footprints make explicit, what kind of details should be modeled and what kind should not. Besides, we expect footprinting to make modelers more confident in the completeness and confinement of their model. The purpose of these experiments is to confirm or to refute this belief. In the next section, we discuss the planning of our experiments.

III. Experiment Planning

The goal of our experiments is to evaluate the effect of model footprinting on the actual and perceived quality of models in the context of students modeling the requirements of a software system. Following the template defined by Wohlin [8], we present the design of the experiments we conducted towards this goal. In the next section, we present the context and the material of the experiments. Then, in Section III-B, we highlight the variables of our experiments and formulate our hypotheses. In Section III-C, we lay out the design of our experiments and we discuss threats to validity in Section III-D.

A. Context and Material

Our experiments were carried out at two sites: Rennes and Zurich. In Zurich, the context of our experiment is a second year (undergraduate level) Software Engineering (SE) course at the University of Zurich. As a prerequisite for this SE lecture, participants had to follow a lecture on modeling[1]. While this modeling lecture does not aim at teaching UML per se, many UML diagrams are covered during the lecture. Students learn the various constructs and modeling guidelines, and they apply this knowledge on some practical exercises. In Rennes, the experiment took place within a Model Driven Engineering (MDE) lecture, which is visited by two second year classes on the graduate level: Miage (oriented towards business) and GL (oriented towards software engineering). The students from Rennes had to visit a lecture on modeling as prerequisite too[2]. At both sites, the experiment was part of a series of compulsory labs and exercises. Students were told that they would not be graded on performance, but that they were expected to solve their modeling assignments in a professional manner to obtain the points assigned to the lab.

Alternatively, we could have recruited these students with some financial incentives. According to Sjøberg et al. [9], our experiment would have been easier to organize with this alternative. However, some students could have refused to participate in our experiment, reducing the number of data points and introducing a bias towards motivated students.

Participants were not aware that we were attempting to evaluate the impact of footprinting, but knew that we would evaluate the quality of their models by comparing them with a reference model. The experiments were monitored by the first author on both sites. He was not involved in any of the courses as an assistant, reducing the bias of the experiment. During the experiment, students were not allowed to talk to each other.

The modeling assignments were solved with MagicDraw[3], an UML modeling tool. We chose MagicDraw over other modeling tools because it fully supports UML class diagrams and state machines and it is intuitive to use. Using a tool makes the modeling task more realistic than pen-and-paper only [9]. It has the additional advantage that it prevents participants from making syntactic mistakes. While the tool is used by the students from Rennes during their curriculum, students from Zurich never used it before the experiment. To overcome this issue, we answered all the questions related to MagicDraw during the experiments. Furthermore, we let students use comments to specify UML elements when they did not find out how to specify them properly with the tools. For example, some students modeled the triggers of state transitions by attaching comments to the transitions. We took these comments into account when assessing the quality of models.

For this experiment, we developed the following material: a modeling assignment with 2 exercises, the reference models and the corresponding static metamodel footprints. This material is included in the appendices. The exercises consist in creating UML models from case descriptions in plain text. The exercises were taken from [10]. The description for the class diagram is the *dental clinic* case (on page 272) and the description for the state machine is the *shipment* case (on page 275). Students were asked to solve each exercise within 45 minutes, so that they had enough time to solve both exercises and fill in the questionnaires during the time allocated for the labs (2 hours). The textbook also provides some solutions, which were used to create the reference models.

To define the static metamodel footprints, we decided not to define (and later analyze) some operations, but rather

[1]This lecture is taught by Prof. Martin Glinz. The description and the material of the module can be found at http://bit.ly/P8RrNZ (in German)

[2]This lecture is taught by Dr. Noël Plouzeau. The description of the module can be found at http://bit.ly/OdIKBU (in French)

[3]http://www.nomagic.com/products/magicdraw.html

derive the static metamodel footprint from the reference model directly. That is, we collected all types and features present in the reference model. We decided to do so to avoid a potential threat to validity: defining model operations ourselves might have introduced some bias in the experiments. However, this decision leaves the validation of the main assumption of footprinting — the purpose of a model can be characterized as a set of operations — out of the scope of this paper. This will be investigated in future work. The experiment remains nevertheless valid, as the origin of the static metamodel footprint is of no importance for modelers.

In total, students answered 4 questionnaires. First, they had to participate in a pre-experimental survey, introducing them to the experiment and collecting their modeling experience. After each modeling exercise, students had to fill in a questionnaire to report how they perceived the modeling task, the case description and the quality of their model. They also had to mention how long they took to perform the modeling task. The post-experimental questionnaire explained briefly the goals of the experiment and invited students to provide us with feedback about the experiments.

B. Variables and Hypotheses

In our experiments, we investigate the impact of footprinting on model quality. We measured 4 quality aspects characterizing the semantic quality of a model as defined by Lindland in [1]: completeness (COM), confinement (CON), correctness (COR) and overall quality (OAQ). Each quality aspect comes in two flavors: actual and perceived quality. Actual quality is the semantic quality of a model with respect to a reference model. We measure this quality by comparing the model to a reference model. Perceived quality is the quality of the model as perceived by its modeler. We measure it with questionnaires. In total, our experiments involve 9 variables: 8 dependent variables and 1 independent variable.

Actual Quality: We quantify actual quality by counting the number of mistakes in the model with respect to a reference model. We distinguish three types of mistakes. A completeness mistake is the absence of some element from the model. For example, in a class diagrams at the analysis stage, the absence of multiplicities or a missing attribute are completeness mistakes. Confinement mistakes are due to the presence of superfluous but correct elements in the model. For example, the presence of interfaces or getter operations are confinement mistakes in an analysis class diagram. All other mistakes are denoted as correctness mistakes. This kind of mistakes includes wrong multiplicities or redundant attributes in a class diagram. ACOM counts the number of completeness mistakes, ACON counts the number of confinement mistakes, ACOR counts the number of correctness mistakes and AOAQ is the total number of mistakes.

Perceived Quality: Perceived quality is measured with a questionnaire. More preciseley, students were asked to what extent they agreed on the following statements using a 5-level Likert scale:

- My model is a good model. (POAQ)
- My model contains all relevant information. (PCOM)
- My model only contains relevant information. (PCON)
- My model only contains valid information. (PCOR)

Method: All participants used the same modeling languages (UML class diagrams and UML state machines). The purpose of the model was stated informally in all assignments. Still, some participants received additional information for one of the two diagrams they had to create: the static metamodel footprint (FP), that is, the set of modeling constructs relevant for the purpose of the model. This is a nominal variable: either the static metamodel footprint was available or not. FP is the only independent variable of our experiment.

Hypotheses: Based on the goal of the experiment, we formulate, for each of the dependent variables, the following null-hypothesis (H_0): there is no difference between the quality of models when the modeler knows (or does not know) the static metamodel footprint (FP). The alternative hypothesis (H_1) is that footprinting has an impact (positive or negative) on the quality of models.

In this paper, we use Wilcoxon rank-sum tests to test our hypotheses. A Wilcoxon rank-sum test is a non-parametric test to compare the median of two samples. We do not use the T-test because the data does not respect all its assumptions: the counts of mistakes (actual quality) follow Poisson distributions rather than normal distributions and the Likert-scales (perceived quality) only deliver ordinal values. If the non-parametric test is not powerful enough, we fit the counts of mistakes to Poisson distributions and compare their parameter.

C. Experiment Design

This experiment only involves one factor whose effect is interesting to us: whether the static metamodel footprint is given in the task assignment or not. However, a nuisance factor may impact the quality of models: the ability of students in modeling. Therefore, we opt for a completely randomized design in which each participant uses both methods (with and without static metamodel footprint) [8]. Overall, our design is inspired by those of Briand et al. when they investigated the benefits of OCL in UML based development [11].

To minimize learning effects between the modeling tasks, we considered two different objects: a class diagram and a state diagram. We keep ordering effects under control by using four groups instead of two, covering all possible permutations. These groups are presented in Table I. For example, participants in group A first model the class diagram without the static metamodel footprint. They then proceed to the state diagram, this time with the static metamodel footprint.

Table I: Experiment design

		Group A	Group B	Group C	Group D
Tasks	1st	Class Diagram With Footprint	Class Diagram Without Footprint	State Machine With Footprint	State Machine Without Footprint
	2nd	State Machine Without Footprint	State Machine With Footprint	Class Diagram Without Footprint	Class Diagram With Footprint

D. Threats to Validity

External Validity: In this experiments, participants are students, half-way to get their Bachelor's or Master's degree. They may not be representative of software analysts working in the field. However, these students know UML at this stage as much as when they will leave the university and enter their professional career. Besides, footprinting is meant to help modelers who do not know much about how their models are used, which is typically the case for novice modelers.

We intentionally kept the modeling tasks small enough so that the students could achieve them within 45 minutes. This constraint reduces the (undesired) effect of fatigue on the results. However, these assignments are not representative of modeling tasks in an industrial context because of their small size and the presence of a complete case description (participants did not have to elicit requirements). Still, if footprinting has an impact on small models, it will likely have an impact on larger models, too. Elicitation was kept out of the experiment, as it is not within the scope of our investigations.

Internal Validity: Besides the availability of the static metamodel footprint, other factors can impact the quality of models, like the ability of participants and ordering effects (learning and fatigue). We kept ability under control by having each participant use both methods, while ordering effects were kept under control by having four groups solving two different modeling tasks in different order.

We provided footprinting in its most basic form: a static metamodel footprint in textual form. Alternatively, we could have provided an example of models containing all constructs from the static metamodel footprint or a tool displaying the model footprint. We chose the most basic form to reduce the bias towards footprinting. Furthermore, the static metamodel footprint can give some hints about completeness, while the static (model) footprint cannot.

Construct Validity: The perceived quality (PCOM, PCON, PCOR and POAQ) is measured using a questionnaire. We phrased our questions so as to avoid bias. Actual quality (ACOM, ACON, ACOR and AOQ) is measured by counting the number of mistakes made with respect to a reference model, that is, the solution of the modeling assignment. This reference model was written by the first author based on the correct model provided in [10]. Still, to mitigate the risk that this domain is biased, the domain was validated by other authors. The count of mistakes is not a fully accurate measure of quality, as some mistakes are more important than others. Still, we did not assign weights to them to keep the measure as objective as possible.

Students may have acted so as to please our expectancies. This threat was reduced by (a) not telling the students the exact hypotheses behind our experiments and (b) having the experiments supervised by a researcher who is not involved in the lectures participants were enrolled to (SE for Zurich, MDE for Rennes). In the next section, we present and discuss the results obtained during the experiments.

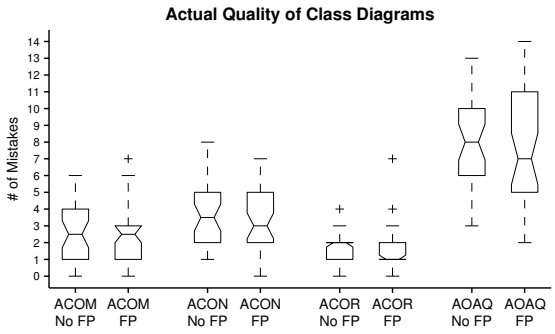
IV. Results Analysis and Interpretation

In total, 86 students participated to our experiments: 14 Miage and 23 GL students from Rennes and 49 SE students from Zurich. From them, we gathered 72 complete datasets including both diagrams and answers to surveys. The datasets are spread almost equally in each group (19 in group A, 15 in B, 19 in C, 19 in D). Based on the pre-experimental survey, most students consider themselves as experienced with modeling and UML in general. However, the average experience is lower with state machines than with class diagrams. Most students found that both the modeling tasks and the case descriptions were clear and that they had enough time to do the modeling. In average, participants took 39 minutes for the class diagram and 32 minutes for the state machine. Footprinting had no impact on the time used for modeling. In the remainder of this section, we first analyze the impact of footprinting on actual quality and then its impact on perceived quality.

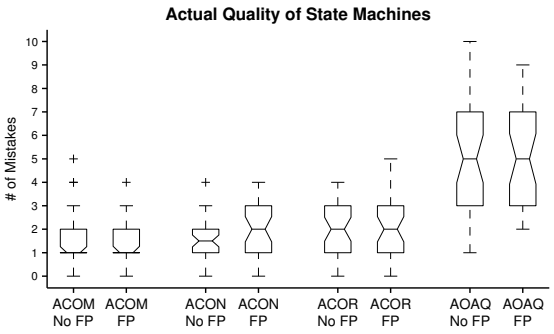
A. Actual Model Quality

To assess the actual quality of models, we compared the models to our reference models. We used more than one reference model, as the information from a textual description may be modeled correctly in different ways. Besides, we ignored differences in names, as long as the names were meaningful. Some other deviations may have been acceptable (e.g., the presence of business operations in a class diagram), but we marked them all to avoid introducing some bias in the results. In the class diagrams, we identified a total of 72 possible mistakes. 31 mistakes are related to confinement, 17 to completeness and 24 to correctness. For the state machines, we have listed 52 mistakes. 16 mistakes are confinement issues, 16 are completeness problems and 20 are correctness mistakes.

The actual quality of models is presented as boxplots in Figure 2. Figure 2a shows the number of mistakes made in class diagrams, while Figure 2b displays the number of mistakes made in the state machines. The results are grouped by the availability of the static metamodel footprint (the treatment) and the various quality aspects (completeness, confinement, correctness and overall quality). The y-axis represents the number of mistakes made. The higher the number of mistakes, the lower the actual quality of a model. In a boxplot, the boxes have lines at the first quartile,



(a) Class Diagrams



(b) State Machines

Figure 2: Actual quality of models

Table II: Median number of mistakes

		ACOM	ACON	ACOR	AOAQ
Class Diagram	No Footprinting	2.50	3.50	2.00	8.00
	Footprinting	2.50	3.00	1.00	7.00
	P-Value	0.91	0.41	0.20	0.63
	Reject H_0?	No	No	No	No
State Machine	No Footprinting	1.00	1.50	2.00	5.00
	Footprinting	1.00	2.00	2.00	5.00
	P-Value	0.62	0.31	0.84	0.76
	Reject H_0?	No	No	No	No

Table III: Confidence interval of λ for the counts of mistakes

		ACOM	ACON	ACOR	AOAQ
Class Diagram	No Footprinting	2.53	3.79	1.74	8.06
		[2.02 - 3.12]	[3.14 - 4.45]	[1.32 - 2.24]	[7.10 - 9.01]
	Footprinting	2.68	3.45	1.55	7.68
		[2.16 - 3.21]	[2.86 - 4.04]	[1.18 - 2.00]	[6.80 - 8.57]
	Altogether	2.61	3.61	1.64	7.86
		[2.24 - 2.98]	[3.17 - 4.05]	[1.34 - 1.93]	[7.21 - 8.51]
State Machine	No Footprinting	1.68	1.50	1.87	5.05
		[1.30 - 2.15]	[1.14 - 1.94]	[1.46 - 2.36]	[4.34 - 5.77]
	Footprinting	1.47	1.76	1.97	5.21
		[1.09 - 1.94]	[1.35 - 2.27]	[1.53 - 2.50]	[4.44 - 5.97]
	Altogether	1.58	1.63	1.92	5.13
		[1.29 - 1.87]	[1.33 - 1.92]	[1.60 - 2.24]	[4.60 - 5.65]

median, and third quartile. The notches around the medians represent their 95% confidence interval, which is useful for comparing medians.

Table II displays the median number of mistakes made. For each kind of diagrams and for each quality attribute, we compare the median number of mistakes made by students who knew the static metamodel footprint with the median number of mistakes made by students who did not. For this comparison, we use a Wilcoxon rank-sum test whose p-value is displayed in Table II. We cannot reject any null-hypotheses, as no p-value is above the usual significance level $\alpha = 5\%$. Thus, we use a parametric model to further investigate whether footprinting had a statistically significant impact on the actual model quality.

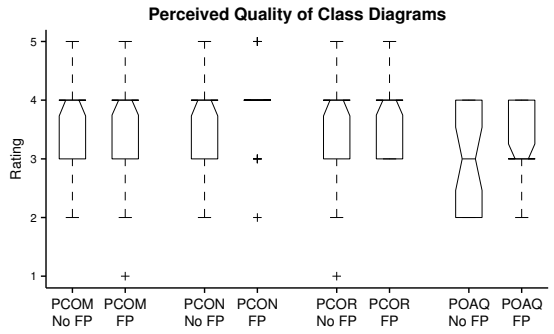
The numbers of mistakes follow Poisson probability distribution laws. Table III displays the average number of mistakes made by students in their models. The average is an unbiased estimator of the λ parameter of the Poisson law. Overall, students made 7.86 mistakes in their class diagrams and 5.13 mistakes in their state machines. Table III also provides the 95% confidence intervals for λ. The difference between the quality with and without footprinting are not statistically significant, as all confidence intervals for λ overlap.

Interestingly, the results obtained for the class diagram contradict those obtained for the state machine. For class diagrams, footprinting improves all quality aspects except completeness. Indeed students with footprinting makes fewer mistakes in total (AOAQ) than without footprinting: fewer confinement mistakes (ACON), fewer correctness mistakes (ACOR) but more completeness mistakes (ACOM). In contrast, footprinting only improves completeness for state machines (ACOM), while degrading all other qualities (ACON, ACOR and AOAQ).

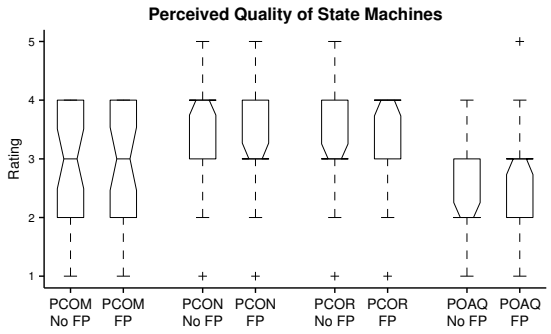
B. Perceived Model Quality

The ratings of perceived quality are plotted as boxplots in Figure 3. Given the ordinal nature of Likert scales, we use the median to characterize the perceived model quality (see Table IV). For class diagrams, footprinting had almost no impact on the perceived quality of models. For state machines, footprinting reduces the perceived confinement (PCON) of models, but improves the perceived correctness (PCOR) and the perceived overall quality (POAQ). However, the p-values of the Wilcoxon rank-sum tests are all above the $\alpha = 5\%$ level required to reject a null hypothesis. Thus, the differences between the medians are not statistically significant.

In general, the perceived quality is coherent with the actual quality. To assess this coherence, we group the models according to their perceived quality and we compute the



(a) Class Diagrams



(b) State Machines

Figure 3: Perceived quality of models

Table IV: Median rating of perceived quality

		PCOM	PCON	PCOR	POAQ
Class Diagram	No Footprinting	4.00	4.00	4.00	3.00
	Footprinting	4.00	4.00	4.00	3.00
	P-Value	0.59	0.22	0.38	0.33
	Reject H_0?	No	No	No	No
State Machine	No Footprinting	3.00	4.00	3.00	2.00
	Footprinting	3.00	3.00	4.00	3.00
	P-Value	0.73	0.28	0.57	0.23
	Reject H_0?	No	No	No	No

Table V: Coherence between perceived and actual model quality

			Actual Quality			
			COM	CON	COR	OAQ
Perceived Quality	Class Diagram	1	5.50 (2)		1.50 (2)	
		2	3.82 (11)	2.67 (3)	2.00 (2)	9.63 (19)
		3	2.70 (20)	3.75 (16)	1.65 (20)	8.08 (25)
		4	2.18 (34)	3.62 (47)	1.68 (44)	6.46 (28)
		5	1.40 (5)	3.67 (6)	1.00 (4)	
	State Machine	1	3.40 (5)	1.67 (3)	2.25 (4)	7.00 (8)
		2	2.33 (21)	1.00 (3)	1.90 (10)	5.71 (28)
		3	1.23 (22)	1.41 (27)	1.75 (24)	4.13 (23)
		4	0.88 (24)	1.67 (33)	2.03 (32)	4.25 (12)
		5		2.67 (6)	1.50 (2)	7.00 (1)

average actual quality for each group. Table V displays these averages. For example, 19 students assessed the overall quality of their model with a 2. The average number of mistakes for these students is 9.63. In comparison, 28 students assessed the overall quality of their class diagram with a 4 and made, in a average, 6.46 mistakes. In most cases, the higher the perceived quality is, the lower the number of mistakes is, and, thus, the higher the actual quality is. We observe similar results for models made with and without static metamodel footprints. Thus, in Table V, we only show the results for all models, no matter whether the static metamodel footprint was available for their construction.

V. Discussion

We have hypothesized that footprinting would improve the quality of models and increase the confidence of the modelers in the quality of their models. After all, footprinting provides additional hints that should have helped the participants in their modeling tasks. However, our experiments did not demonstrate any statistically significant effect of footprinting on model quality. Unfortunately, our post experimental survey did not include any question to further explain this issue. Thus, more research is needed to investigate whether this is due to the design of our experiments. In this section, we propose and discuss possible explanations, providing some direction for future research.

The modeling assignements may have been too simple to demonstrate the benefits of footprinting. Indeed, the case descriptions do not include many irrelevant details with respect to the given modeling purpose. In the post-experimental surveys, most participants disagreed that both case descriptions were too detailed. Thus, the modelers had no difficulty to figure out what was to be modeled, no matter whether or not they knew the static metamodel footprint. The results may have been different if we had used a large case description with many superfluous details. In such a case, the static metamodel footprint may help for deciding which details should be included in the model and which details should be left out.

Participants may have overlooked the static metamodel footprint or may not know how to exploit this information. Indeed, some kinds of mistakes would have been avoided if the static metamodel footprint had been used properly. The number of students committing these mistakes supports this explanation. For example, 30 students forgot to model the triggers on transitions in the state machines: 18 students had the static metamodel footprint, 12 had not. For the class diagrams, 11 students modeled realization relationships, which are superfluous for the purpose at hand: 8 of them had the static metamodel footprints, 3 had not. This also suggests

that our measure for actual quality is not responsible for the results.

Future research should clarify to what extent the form of footprinting reduces its impact on model quality. We chose to provide the static metamodel footprint as text, directly integrated in the modeling task without highlighting it. Alternatively, we could have extended the modeling tool to provide the students with some feedback about the model footprint, displaying warning messages about superfluous or missing elements. We could also have presented an example of a model containing all the constructs in the static metamodel footprint. In these forms, the participants can more easily use the information provided by footprinting than in its basic form and they can less easily overlook it.

In our experiments, we did not introduce footprinting and we did not train participants to use it. Thus, participants may not be experienced enough in metamodeling or in the UML metamodel to properly exploit the static metamodel footprint. While this may be the case for the students in Zurich, these subjects were taught to the students from Rennes in the MDE lecture. Yet, there is no significant difference in the results from both sites. Besides, most students considered themselves as experienced in modeling, in UML and in class diagrams during the pre-experimental survey. Still, footprinting may require some training before it can actually deliver some benefits to the modelers.

VI. Conclusion and Future Work

Creating requirement models is not an easy task. In addition to eliciting information about the original, modelers need to understand the purpose of their model. Often, this purpose is kept implicit and the only indication is a modeling language. In a MDE setting, where the model is often used to feed some model operations (like queries or transformations), footprinting can be used to assess and improve the quality of models. Still, the benefits of footprinting on model quality were not yet empirically investigated.

To investigate the benefits of footprinting, we conducted a pair of controlled experiments involving students from two universities, Rennes and Zurich. Participants had to model a class diagram and a state machine. Some participants used footprinting for the class diagram, while others used it for the state machine. We evaluated both the actual quality of models — by counting the number of mistakes with respect to a reference model — and the quality perceived by the modeler — by using a questionnaire.

Our results are inconclusive: the effect of footprinting on quality is not statistically significant and the results in the class diagram contradict those in the state machine. We believe that these results can be accounted by the way we presented footprinting to the participants: a static metamodel footprint in text form. Participants may have overlooked it or may not have used it properly. We would have obtained different results if we had trained the participants to footprinting or if we had provided them with a modeling tool displaying feedback on the confinement and the completeness of their models.

Further research is needed for establishing the impact of footprinting on model quality. Furthermore, footprinting is not only meant for creating better models, it can also be used to better understand model operations. Thus, investigating the impact of footprinting on the comprehension of an operation is another direction for demonstrating the benefits of footprinting empirically. Finally, one could evaluate to what extent a set of operations is a good characterization of a model's purpose.

Acknowledgment

Our work is partially funded by the Swiss National Science Foundation under the project 200021_134543 / 1. We are grateful to Irina Todoran and Clément Guy who helped on the organisation of the experiments and also the students from Rennes and Zurich who participated in the experiments.

References

[1] O. I. Lindland, G. Sindre, and A. Sølvberg, "Understanding quality in conceptual modeling," *IEEE Software*, vol. 11, no. 2, pp. 42–49, 1994.

[2] C. Jeanneret, M. Glinz, and B. Baudry, "Estimating footprints of model operations," in *33rd International Conference on Software Engineering (ICSE 2011)*, Waikiki, Honolulu, HI, USA, 2011, pp. 601–610.

[3] J. Krogstie, G. Sindre, and H. Jorgensen, "Process models representing knowledge for action: a revised quality framework," *European Journal of Information Systems*, vol. 15, no. 1, pp. 91–102, 2006.

[4] D. L. Moody, G. Sindre, T. Brasethvik, and A. Sølvberg, "Evaluating the quality of information models: Empirical testing of a conceptual model quality framework," in *25th International Conference on Software Engineering (ICSE '03)*, 2003, pp. 295–305.

[5] S. España, N. Condori-Fernandez, A. Gonzalez, and Ó. Pastor, "Evaluating the completeness and granularity of functional requirements specifications: A controlled experiment," in *17th International Conference on Requirements Engineering (RE'09)*, Atlanta, GA, USA, 2009, pp. 161–170.

[6] P. Mohagheghi, V. Dehlen, and T. Neple, "Definitions and approaches to model quality in model-based software development: A review of literature," *Information and Software Technology*, vol. 51, no. 12, pp. 1646–1669, 2009.

[7] C. Jeanneret, M. Glinz, and T. Baar, "Modeling the purposes of models," in *Modellierung 2012*, Bamberg, Germany, 2012, pp. 11–26.

[8] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[9] D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, E. F. Koren, and M. Vokác, "Conducting realistic experiments in software engineering," in *1st International Symposium on Empirical Software Engineering (ISESE 2002)*, 2002, pp. 17–26.

[10] J. W. Satzinger, R. B. Jackson, and S. D. Burd, *Systems Analysis and Design in a Changing World*. Boston, MA, USA: Course Technology, 2008.
[11] L. C. Briand, Y. Labiche, H. D. Yan, and M. Di Pent, "A controlled experiment on the impact of the object constraint language in UML-based development," in *20th International Conference on Software Maintenance (ICSM 2004)*, Chicago, IL, USA, 2004, pp. 380–389.

APPENDIX A.
DENTAL CLINIC

Instructions

The following section describes a system to manage patient records in a dental clinic. Your task is to produce an analysis model of this system with a class diagram based on the description. Your model will be used to generate the skeleton of a glossary. We will compare your model to a reference model: Make sure to include every relevant piece of information, but not more!

Static Metamodel Footprint

Note: The static metamodel footprint has not been handed to all participants.

In this model, we are only interested in the following items:

- Classes
  - name of the class
- Attributes
  - name of the attribute
- Associations
- Association Ends
  - multiplicity of the association end
- Generalizations

Case Description

A clinic with three dentists and several dental hygienists needed a system to help administer patient records. This system does not keep any medical records. It only processes patient administration.

Each patient has a record with his/her name, date of birth, gender, date of first visit, and date of last visit. Patient records are grouped together under a household. A household has attributes such as name of head of household, address, and telephone number. Each household is also associated with an insurance carrier record. The insurance carrier record contains name of insurance company, address, billing contact person, and telephone number.

In the clinic, each dental staff person also has a record that tracks who works with a patient (dentist, dental hygienist, x-ray technician). Because the system focuses on patient administration records, only minimal information is kept about each dental staff person, such as name, address, and telephone number. Information is maintained about each office visit, such as date, insurance copay amount (amount paid by the patient), paid code, and amount actually paid. Each visit is for a single patient, but, of course, a patient will have many office visits in the system. During each visit, more than one dental staff person may be involved in the visit by doing a procedure. For example, the x-ray technician, dentist, and dental hygienist may all be involved in a single visit. In fact, some dentists are specialists in such things as crown work, and even multiple dentists may be involved with a patient. Detailed information is kept about procedures performed by a staff person during a visit. This information includes type of procedure, description, tooth involved, the copay amount, the total charge, the amount paid, and the amount insurance denied.

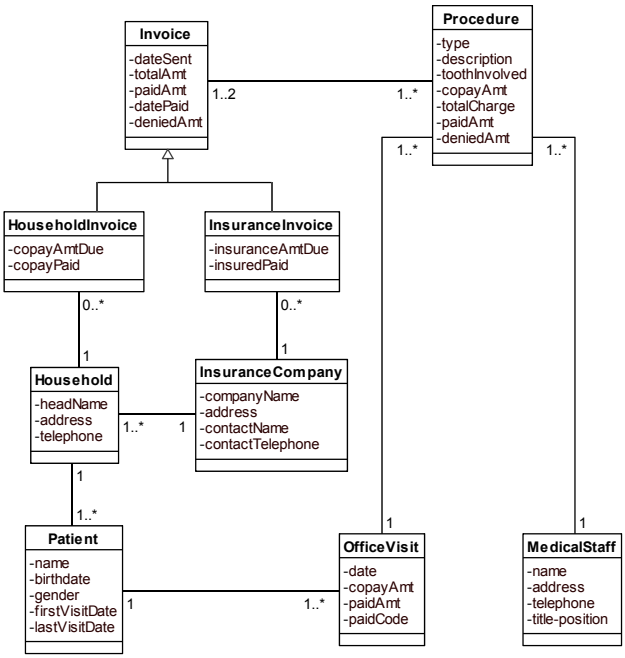


Figure 4: Reference model for the dental clinic domain

Finally, the system also keeps track of invoices. There are two types of invoices: invoices to insurance companies and invoices to heads of household. Both types of invoices are fairly similar, listing each visit, the procedures involved, the patient copay amount, and the total due. Obviously, the totals for the insurance company are different from the patient amounts owed. Even though an invoice is a report (printed out), it also maintains some information such as date sent, total amount, amount already paid, amount due and also the total received, date received, and total denied. (Insurance companies do not always pay all they are billed.)

Reference Model

A reference model is given in Figure 4.

APPENDIX B.
SHIPMENT

Instructions

The following section describes the behavior of a shipment by Union Parcel Shipments. Your task is to document this behavior with a state machine. Your model will be used to generate an implementation based on the *State* design pattern. We will compare your model to a reference model: Make sure to include every relevant piece of information, but not more!

Static Metamodel Footprint

Note: The static metamodel footprint has not been handed to all participants.

In this model, we are only interested in the following items:

- States
  - name of the state
- Transitions
  - trigger of the transition (as signal event)
- Signal Events
  - name of the event
- Initial States
- Final States

Case Description

A shipment is first recognized after it has been picked up from a customer. After it is in the system, it is considered to be active and in transit. Every time it goes through a checkpoint, such as arrival at an intermediate destination, it is scanned, and a record is created indicating the time and place of the checkpoint scan. The status changes when it is placed on the delivery truck. It is still active, but now it is also considered to have a status of delivery pending. Of course, after it is delivered, the status changes again.

From time to time, a shipment has a destination that is outside the area served by Union. In those cases, Union has working relationships with other courier services. After a package is handed off to another courier, it is noted as being handed over. In those instances, a tracking number for the new courier is recorded (if it is provided). Union also asks the new courier to provide a status change notice after the package has been delivered.

Unfortunately, from time to time a package gets lost. In that case, it remains in an active state for two weeks but is also marked as misplaced. If after two weeks the package has not been found, it is considered lost. At that point, the customer can initiate lost procedures to recover any damages.

Reference Model

A reference model is given in Figure 5.

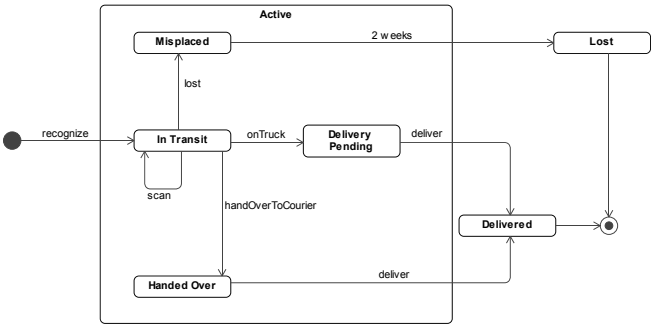


Figure 5: Reference model for the behavior of a shipment