T. Sağlam, L. Schmid, S. Hahner, and E. Burger, "How Students Plagiarize Modeling Assignments," in 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), ACM/IEEE, 2023, doi: 10.1109/MODELS-C59198.2023.00032

How Students Plagiarize Modeling Assignments

Timur Sağlam, Larissa Schmid, Sebastian Hahner, Erik Burger
KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
firstname.lastname@kit.edu

***Abstract*—Plagiarism is a prevalent issue in computer science education, with students employing various techniques to conceal their plagiarism. While it is essential to understand how students engage in plagiaristic behaviors, it remains unclear how students do so for modeling assignments. To close this gap, we conducted an experiment involving novice modelers, instructing them to plagiarize a modeling assignment while concealing its source. Thus, this paper explores students' specific techniques to conceal modeling plagiarism. Our work initiates a discussion on plagiarism in modeling assignments and sheds light on the challenges associated with detecting and addressing it.**

***Index Terms*—Plagiarism Detection, Modeling Plagiarism, Plagiarism Obfuscation, Modeling Assignments, MDE Education**

I. Introduction

Plagiarism is a significant concern in educational settings, especially in today's computer science education, as digital deliverables can be easily copied and altered [1, 2, 3]. Plagiarism is particularly prevalent when assignments are mandatory, for instance, in introductory-level courses [4]. Students demonstrate creativity in concealing plagiarism, particularly regarding digital deliverables such as source code or models. They employ *obfuscation* techniques such as renaming, reordering, or restructuring to make the plagiarism less recognizable and more challenging to detect [5, 6]. Despite preventive measures and educational interventions, plagiarism in assignments persists. To make matters worse, online resources, e.g. repositories of solved exercises, further simplify plagiarism [7]. To better recognize and prevent plagiarism, it is essential to understand how students approach and engage in plagiaristic behaviors.

Regarding programming assignments, the forms of plagiarism are well researched [5]. This includes classifications of code obfuscation techniques [8, 6] and the automated detection of plagiarism in student submissions [9]. However, only limited research exists on plagiarism in modeling assignments. Furthermore, modeling assignments are susceptible to plagiarism since they are complex and require domain understanding and problem-solving skills [7]. Yet, these assignments are often the only way to evaluate the student's performance [10]. While plagiarism detection for modeling assignments has been proposed recently [7, 11], it remains unclear how students engage in plagiaristic behaviors in modeling assignments. Related work from the domain of modeling clone detection [12] does not address this issue: Plagiarism detection involves an attacker scenario, where the plagiarizer intentionally tries to conceal the plagiarism by employing various obfuscation techniques. In contrast, clone detection focuses on identifying identical or similar sections in models without considering deliberate obfuscation by plagiarizers. Therefore, the approaches and techniques developed for modeling clone detection cannot be directly applied to the context of modeling plagiarism. In essence, the lack of knowledge about students' plagiaristic behaviors hinders detection efforts.

This paper addresses this issue by systematically analyzing how students plagiarize and obfuscate in modeling assignments. We present the results of an experiment where ten novice modelers were asked to plagiarize a given metamodeling assignment's solution and conceal their plagiarism, i.e., hiding the relation to the original. We employed a mixed-method approach, combining quantitative analysis of the plagiarized metamodels and qualitative data from the participants' descriptions of how they tried to conceal the plagiarism. We show how frequently different techniques were employed and to what level the models were altered. In the experiment's result evaluation, we asked the course instructors to inspect both plagiarized and original solutions. Additionally, we applied a state-of-the-art plagiarism detector [11] to test the automatic plagiarism detection quality. In sum, we address and contribute to the following research questions:

RQ1: How do students conceal modeling plagiarism?
RQ2: Can modeling plagiarism be detected by the instructors?
RQ3: Can modeling plagiarism be detected automatically?

The experiment shows how the ten participants engage in modeling plagiarism and how they described their approach (**RQ1**). We also report on our findings from both the manual inspection (**RQ2**) and the automated analysis (**RQ3**). We publish all pertinent data in the supplementary material. The results show that the participants employ various obfuscation techniques, with the renaming of elements occurring most frequently, followed by re-ordering elements. On average, students alter every second element of the modeling assignment. Nevertheless, human and tool-based inspection can still detect these plagiarism instances. However, manual inspection quickly becomes impractical with large course sizes. This paper contributes to a better understanding of plagiarism in modeling assignments by investigating students' obfuscation techniques. Ultimately, we provide insights to enhance fairness and academic integrity in assessing modeling assignments.

This work is based on the project *SofDCar* (19S21002), which the German Federal Ministry for Economic Affairs and Climate Action funds. It is also supported by the Ministry of Science, Research and the Arts Baden-Württemberg (Az: 7712.14-0821-2), by "Kerninformatik am KIT" funded by the Helmholtz Association (HGF), and the topic Engineering Secure Systems of the HGF and by KASTEL Security Research Labs.

Table I: Obfuscation techniques employed by the participants, classified according to Babur et al. [12]. We include how often each technique occurred across all participants (denoted as *#Occ*) and how many participants employed it (denoted as *#Part*).

Techniques	Description	#Occ.	#Part.	Type
Cosmetic Renaming	Changing capitalization of names, introducing or resolving typos.	1	1	A
Minor Renaming	Introducing or resolving abbreviations, adding and removing suffixes or prefixes.	85	9	B
Major Renaming	Changing names to synonyms, translations, or entirely different names.	141	7	C
Re-Order Features	Re-ordering attributes and references of a classifier.	10	1	B
Re-Order Package	Re-ordering the elements contained within a single package.	57	4	B
Re-Order Classifiers	Moving classifiers from one package to another (re-ordering across the model).	1	1	C
Introduce Package	Create a package and add existing elements from other packages or other packages.	20	3	C
Dissolve Package	Deleting a package and moving its contained elements to other packages.	8	4	C
Delete Feature	Delete an existing attribute, relation, or operation of a classifier.	26	7	B
Delete Classifier	Delete an existing classifier from the model.	11	6	C
Delete Package	Delete an existing package from the model (can be part of dissolving a package).	5	4	B
Insert Feature	Insert a new attribute, relation, or operation into a classifier.	6	4	B
Insert Classifier	Insert a new classifier into the model.	10	3	C
Insert Package	Inserting a new package into the model (may be combined with moving elements).	5	4	B
Change Property	Changing an element's property, e.g., *abstract* for classifiers, *ordered* or the cardinality for references.	31	3	B
Remove Inheritance	Remove inheritance relation between the two classifiers.	4	1	C
Add Inheritance	Add inheritance relation between the two classifiers.	7	2	C
Change Inheritance	Changing the inheritance hierarchy structurally without changing it semantically.	9	2	D

II. Experiment Design

We conducted an experiment with novice modelers, where they plagiarized an EMF [13] metamodel and obfuscated the relation to the original metamodel. In total, ten students voluntarily participated in the experiment. The experiment consisted of two tasks conducted sequentially. First, the participants were asked to copy and modify a given source metamodel to conceal the plagiarism while still fulfilling the given assignment. Second, we asked the participants for a brief description outlining their techniques to disguise the act of plagiarism and its relation to the original solution. The duration of the experiment was 30 minutes. Since participants used their own solutions as a basis for plagiarism, they did not require additional familiarization. *1) Scenario:* We instructed the participants to imagine a scenario where a deadline for a modeling assignment was coming up. However, they had access to a solution provided by a hypothetical classmate and thus decided to plagiarize it. To that end, the participants were asked to conceal the plagiarism from the instructor of the hypothetical course while still fulfilling the assignment. *2) Participants:* We conducted our experiment with ten students from a practical course on model-driven software development. It is a master's level computer science elective course at Karlsruhe Institute of Technology, Germany. The students are novice practitioners of model-driven engineering and had little to no prior metamodeling experience before attending the course. The course covers typical [10] topics like (meta)modeling and model transformations. *3) Metamodeling Assignment:* Our experiment uses a typical [10] modeling assignment [14] of an MDE practical course, which tasks the students with creating an EMF metamodel for designing component-based system architectures. The metamodel includes four different architectural views: The component repository, the component assembly, the system's hardware environment, and the components' allocation. Students usually solve this modeling assignment in groups, ranging between two and five students. In the past, students' solutions for this assignment contained, on average, 5 packages, 39 classifiers, 45 references, 10 attributes, and 1 operation [11]. 5 *4) Ethical Considerations:* During this experiment, ethical considerations were given due attention. Participants voluntarily partook and were fully informed about the experiment's scope and purpose. They explicitly agreed to use and publish their artifacts for research purposes, ensuring confidentiality by anonymizing them.

III. Results

In this section, we present our findings [14], addressing **RQ1** by analyzing participants' obfuscation techniques, **RQ2** by examining human plagiarism detection effectiveness, and **RQ3** by evaluating an automated tool's detection performance.

A. Obfuscation Techniques

To analyze the obfuscation techniques employed by the students (**RQ1**), we first used *EMF Compare* to identify potential differences between the plagiarized models and their originals. Unsurprisingly, in some cases, EMF Compare did not accurately produce the correct differences [15], highlighting the limitation of relying solely on model differencing for plagiarism detection. In the second step, we thus manually inspected the models, carefully examining them for potential obfuscation. To ensure the validity of our observations, we cross-referenced our findings with the textual descriptions provided by the students. In our experiment, the participants engaged in a range of individual alterations to obfuscate the plagiarism, with the number of alterations varying between 10 and 91. Thereby we count semantic alterations, e.g. moving an element is a single one. The mean number of alterations is 43.7, while the median is 31.5. These results imply an average of approximately one alteration for every two model elements in the source model.

We classify the participant's techniques according to the modeling clone types from Babur et al. [12], based on the UML clone types of [16]. While designed for clones rather than

plagiarism, this classification enables systematic categorization of plagiarism techniques. Babur et al. [12] describe **Type-A** clones as exact duplicates except for layout, formatting, internal IDs, and purely cosmetic name changes. **Type-B** clones have minor syntactic and semantic changes to names, types, attributes, and minor additions or removals. **Type-C** clones include more extensive alterations, additions, and removals of elements and their properties. Finally, **Type-D** clones are semantically equivalent with different structures and content.

We provide an overview of the techniques, indicating the frequency of their application and the number of participants who utilized each technique. As summarized in Table I, the participants employed various obfuscation methods during the experiment. The types of changes can be described as follows: The most common method was renaming, accounting for about half of the total changes made by all participants. Notably, the participants tended to utilize sophisticated renaming strategies, resulting in the majority of these alterations being categorized as *major renaming* and none as *cosmetic renaming*. As an example of such major renaming, one participant renamed a classifier from ”Component” to ”BuildingBlock.” Other properties, such as *abstract* or *ordered*, were also subject to changes, albeit less frequently than names.

Another common practice observed was the reordering of elements. However, most participants refrained from moving classifiers across packages. Instead, they focused on rearranging the contents of the packages themselves or altering the order of features within the classifiers. Some participants also introduced or dissolved packages, thus moving the contained packages or classifiers. The insertion and deletion of elements were far less common than expected. While in source code assignments, the insertion of lines is one of the most prevalent obfuscation techniques [5], our findings did not reflect this for modeling assignments. Deleting elements, particularly features, was observed slightly more frequently than insertions. Four participants also modified the inheritance relations between classifiers. Interestingly, two students went beyond simple alterations and completely replaced the inheritance hierarchy with a different one while ensuring that the underlying semantics remained intact. This is the only technique we classified as *Type D* (semantically equivalent, structurally different). In summary, Table I shows that students employ diverse obfuscation techniques with varying frequencies.

B. Human Detectability

To evaluate if humans can detect the participants' plagiarism instances (**RQ2**), we asked the course instructor to review eight metamodels regarding their originality and whether they would accept them as valid solutions. Due to time constraints, we only used a subset of the metamodels. To include weaker and stronger instances of plagiarism, we selected the plagiarism instances based on the number of alterations made by the plagiarizers. The subset for the review consists of four original solutions. It also contains three plagiarism instances derived from one solution and one instance from another.

For the review, we used the *Think Aloud* method, asking the instructor to verbalize what they were thinking and doing. The instructor reviewed the models one after another, arranging them side-by-side and first checking for their overall structure and partition into packages. Within packages, they checked the order of elements and the naming of the elements, focusing on the data types that had to be modeled as part of the assignment. They also inspected the OCL constraints contained for the models that seemed similar. Within 20 minutes, they were able to identify all plagiarism clusters correctly. One of the plagiarized metamodels would not have been accepted as a valid solution as it is a poor translation of the original, resulting in nonsensical names. While the instructor identified all plagiarism instances correctly in this experiment, they also stated that this was possible only due to the small sample size. For more metamodels, they would be unable to compare them all manually side-by-side but would employ a tool.

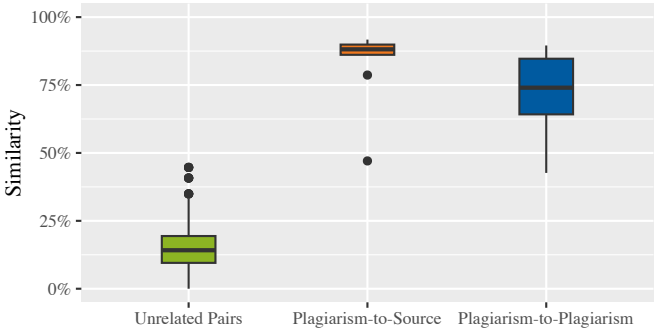

Figure 1: Similarities computed by JPlag for unrelated pairs, plagiarisms with their sources, and plagiarisms of the same source.

Type	Med	Mean	Min	Max	Q1	Q3
Unrelated Pairs	14.79	15.52	0.00	51.15	10.08	20.78
Plag.-to-Source	89.27	84.71	47.06	95.31	86.35	91.58
Plag.-to-Plag.	77.46	72.55	42.65	92.79	64.52	85.36

Table II: Similarity metrics for the pair types in Figure 1, a higher difference to the unrelated pairs means better detection.

C. Tool-based Detectability

To answer whether the plagiarized models can still be detected with a state-of-the-art tool (**RQ3**), we employed the token-based plagiarism detector JPlag [9, 11]. To create a comprehensive labeled dataset, we combined the plagiarism instances from ten participants, along with their corresponding sources, and included unrelated solutions from previous years. This resulted in a dataset of 31 solutions, which we analyzed using JPlag. JPlag computes similarities for each assignment pair, resulting in 465 pairs, which we depict in Figure 1. Detecting the relationship between two instances of plagiarism from the same source is significantly more challenging than the relationship between a plagiarism instance and its source; as for the former case, the alterations made by both plagiarizers accumulate, leading to plagiarism up to twice as well obfuscated.

The plagiarism-to-plagiarism pairs exhibit considerably lower similarities, with a median value of 77.5 percent, in contrast to the 89.3 percent for the plagiarism-to-source pairs. Furthermore,

except for one outlier, all plagiarism-to-source pairs exhibit higher similarity values than all plagiarism-to-plagiarism pairs. This outlier is the plagiarism instance by one participant, who achieves the reduced technique with two techniques. First, they were thorough, making 91 individual changes to the source model, corresponding to one change per model element. Second, they heavily relied on re-ordering model elements, a weakness of JPlag. Nonetheless, the similarity calculated by JPlag is high enough to detect this outlier. Both groups, however, can be clearly distinguished from the pairs of unrelated models, which exhibit a significantly lower median similarity value of 14.8 percent. In summary, our findings show that modeling plagiarism can be detected automatically.

IV. Discussion

Lessons Learned: Our experiment revealed that despite the various types of changes, most students predominantly utilized renaming, deletion, and reordering as obfuscation techniques. This highlights the limitations of relying on names for plagiarism detection. In modeling, names hold significant importance compared to code, but they enable effortless obfuscation attacks for plagiarism. Similarly, unique identifiers cannot be solely relied on, as they can be manipulated, hindering the identification of plagiarized elements. Detection tools should be designed to minimize the impact of easily manipulable characteristics of modeling artifacts. Students rarely utilized type A changes, indicating their intention to refrain from employing overt modifications. They also utilized very few type D changes, which might be more common among experienced modelers. We observed that the relation of two plagiarism instances of the same original is significantly harder to detect than the relations of the plagiarisms to the original. Although modeling plagiarism can still be detected by humans or via tools, it may become more challenging with increased time spent obfuscating or with more experienced students. In our experiment, the instructor often relied on model elements overlooked during the participant's obfuscation attempts, such as OCL constraints. Furthermore, they also noted that manual comparison would be impractical for larger sample sizes, necessitating using a plagiarism detection tool. These findings emphasize the need for continuous improvement of tool-based detection methods.

Limitations: Our study was conducted in a simulated setting, so we did not directly observe natural instances of plagiarism. Thus, the effort by the students may be less pronounced than in a real setting. Nonetheless, we maintain that these conditions are adequate for observing *how* students plagiarize. Furthermore, motivating students to participate in experiments presents a challenge. To address this, we kept the duration of the experiment relatively short. Nevertheless, the number of participants could be higher, which could impact the generalizability of our findings. For the experiment, we employed a single modeling assignment. However, a broader sample, including assignments of varying complexity, could potentially expose further plagiarism strategies. Additionally, exploring behavioral modeling languages like sequence diagrams would be of interest. Despite this limitation, the experiment provides valuable insights into the methods employed by students in plagiarizing modeling assignments.

V. Conclusion

We examined how students engage in plagiaristic behaviors for modeling assignments. By conducting an experiment with ten novice modelers, we gained insights into the specific obfuscation techniques used. Our findings highlight the importance of understanding modeling plagiarism, as its detection is a significant challenge at scale. Our work contributes to the ongoing discussion on academic integrity and provides valuable insights to better recognize plagiarism. In future work, we aim to explore automated obfuscation and plagiarism generation.

References

[1] G. Cosma and M. Joy, "Towards a definition of source-code plagiarism," *IEEE Transactions on Education*, vol. 51, no. 2, pp. 195–200, May 2008.

[2] W. Murray, "Cheating in computer science," *Ubiquity*, vol. 2010, p. 2, 06 2010.

[3] T. Le, A. Carbone, J. Sheard, M. Schuhmacher, M. de Raath, and C. Johnson, "Educating computer programming students about plagiarism through use of a code similarity detection tool," in *LaTiCE*, Mar. 2013.

[4] C. Park, "In other (people's) words: Plagiarism by university students–literature and lessons," *Assessment & Evaluation in Higher Education*, vol. 28, no. 5, pp. 471–488, Oct. 2003.

[5] M. Novak, M. Joy, and D. Kermek, "Source-code similarity detection and detection tools used in academia: A systematic review," *ACM Transactions on Computing Education*, vol. 19, no. 3, pp. 1–37, Sep. 2019.

[6] O. Karnalim, "Detecting source code plagiarism on introductory programming course assignments using a bytecode approach," in *ICTS*, 2016, pp. 63–68.

[7] S. Martínez, M. Wimmer, and J. Cabot, "Efficient plagiarism detection for software modeling assignments," *CSE*, vol. 30, no. 2, pp. 187–215, Jan. 2020.

[8] J. Faidhi and S. Robinson, "An empirical approach for detecting program similarity and plagiarism within a university programming environment," *Computers & Education*, vol. 11, no. 1, p. 11–19, Jan. 1987.

[9] L. Prechelt, G. Malpohl, M. Philippsen *et al.*, "Finding plagiarisms among a set of programs with jplag." *J.UCS*, vol. 8, no. 11, p. 1016, 2002.

[10] F. Ciccozzi, M. Famelis, G. Kappel, L. Lambers, S. Mosser, R. F. Paige, A. Pierantonio, A. Rensink, R. Salay, G. Taentzer, A. Vallecillo, and M. Wimmer, "How do we teach modelling and model-driven engineering? a survey," in *MODELS Proceedings*, New York, NY, USA, Oct. 2018, p. 122–129.

[11] T. Sağlam, S. Hahner, J. W. Wittler, and T. Kühn, "Token-based plagiarism detection for metamodels," in *MODELS-C Proceedings*, New York, NY, USA, 2022, p. 138–141.

[12] O. Babur, L. Cleophas, and M. van den Brand, "Metamodel clone detection with SAMOS," *COLA*, vol. 51, Apr. 2019.

[13] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed., 2009.

[14] T. Sağlam, L. Schmid, S. Hahner, and E. Burger, "Supplementary Material for How Students Plagiarize Modeling Assignments," Aug. 2023, https://doi.org/10.5281/zenodo.8281291.

[15] J. W. Wittler, T. Sağlam, and T. Kühn, "Evaluating model differencing for the consistency preservation of state-based views," *JOT*, vol. 22, no. 2, pp. 2:1–14, July 2023, ECMFA'23.

[16] H. Störrle, *Effective and Efficient Model Clone Detection*, Cham, 2015, pp. 440–457.