A Proposal for More Realistic Case Descriptions in Modeling Exercises

Pierre Maier
University of Duisburg-Essen
Essen, Germany
pierre.maier@uni-due.de

Vicky Kadziolka
University of Duisburg-Essen
Essen, Germany
vicky.kadziolka@uni-due.de

Abstract

Modeling education relies on modeling exercises that are constituted by case descriptions. Many case descriptions explicitly state all required abstractions, reducing modeling activity largely to the translation of a natural-language text into the constructs of a modeling language. This paper proposes a set of five guidelines, abbreviated by the acronym L.I.C.I.A., for formulating more realistic case descriptions that expose students to modeling decisions more commonly encountered in practice. We apply the guidelines for formulating an example case description tailored to modeling a UML class diagram. Our first experiences with this case description suggest that case descriptions following the L.I.C.I.A. guidelines confront students with additional modeling decisions, such as choosing the data type of an attribute based on example values. Future work should investigate the guidelines for case descriptions in various kinds of modeling exercises and conduct a more comprehensive evaluation of using case descriptions following the guidelines in different teaching contexts.

CCS Concepts

• **Software and its engineering** → *Unified Modeling Language (UML)*; **Object oriented development**; **Abstraction, modeling and modularity**.

Keywords

modeling education, case descriptions, modeling exercises, object-oriented modeling

ACM Reference Format:
Pierre Maier and Vicky Kadziolka. 2026. A Proposal for More Realistic Case Descriptions in Modeling Exercises. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3837062.3838916

1 Introduction

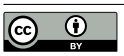
A typical part of modeling education, especially in teaching object-oriented modeling (OOM), is the use of modeling exercises, as also recommended by many educators in the fields [e.g., 2, 4, 9]. Modeling exercises are applied as a teaching instrument to support problem-oriented learning of students. They may also be used to evaluate whether students have acquired the mandatory learning objectives and can be, as such, part of exams. Modeling exercises are also a crucial part of empirical studies on students' modeling competencies and processes [e.g., 22] and, more recently, studies investigating the potential of using large language models (LLMs) to automatically solve modeling exercises [e.g., 6, 10].

Conceptual modeling, in general, requires students to construct abstractions over a domain. In modeling exercises, required domain information is typically provided via descriptions of a case that belongs to a specific domain. Although these case descriptions are an essential constituent of the modeling problem, they are rarely separately discussed in publications. To that end, it may be surprising to find that the case descriptions used in various studies are very similar in the structure of their formulations. Consider the following examples:

> **Example 1** "All employees of the car rental company are assigned an alphanumeric personnel number, and their first name and last name, date of birth and hire date are recorded. Employees can have one supervisor." [21, p. 1007]
>
> **Example 2** "In each play, one or more performers can participate, who are identified by their name." [7, web appendix]

In each example, the domain concepts are already polished in a way that allows for direct and unadjusted adoption in a modeling solution. Considering example 2, two classes `Play` and `Participant` can be added and related by a `participates_in` association. All required labels and information are stated explicitly and completely. The main focus of exercises using such case descriptions is the translation of natural-language statements into a required modeling language; in the case of OOM typically UML. While this is an important aspect of OOM education, students learning OOM must also acquire the competence to construct high-quality abstractions [9]; only translating one representation into another is not enough. A similar critique of case descriptions in modeling exercises is voiced by Engels et al. [9].

Against this background, we propose a set of guidelines for formulating case descriptions that foster abstraction-oriented modeling activities instead of reducing modeling exercises to translation activities. To the best of our knowledge, this presents the first comprehensive proposal of guidelines for formulating case descriptions in modeling exercises. The guidelines are based on plausible assumptions about available information in practical modeling settings. They are also informed by other researchers' comments on OOM education. It is important to note that the guidelines presented here serve as an initial proposal only and are intended to spark further discussions about designing case descriptions for modeling exercises. We have not conducted empirical studies on real-world modeling practice to infer the guidelines. They should thus not



This work is licensed under a Creative Commons Attribution 4.0 International License.
MODELS Companion 2026, Málaga, Spain
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2903-4/26/10
https://doi.org/10.1145/3837062.3838916

be misinterpreted as 'complete' or 'correct' in the sense that they accurately reflect empirically observed modeling practice.

The paper is structured as follows. In Sec. 2 we present and justify our proposed guidelines for case descriptions in modeling exercises. We applied these guidelines to formulate an example case description as part of UML model creation exercises we used in our undergraduate course on enterprise modeling. We present the example case description and outline our teaching context in Sec. 3. We collected students' submissions on this modeling exercise over two semesters. In Sec. 4 we report on the main anomalies we observed in the submitted solutions and discuss how they relate to each proposed guideline. Finally, Sec. 5 concludes the paper.

2 L.I.C.I.A. Guidelines for Formulating Case Descriptions

We have formulated five guidelines for formulating case descriptions that are abbreviated by the acronym L.I.C.I.A. Each guideline introduces challenges to modeling exercises that we think students should be confronted with.

Guideline 1 – ***Label ambiguity****: The description should include ambiguous references to domain concepts.* Case descriptions for modeling exercises are often characterized by a consistent naming scheme that refers to domain concepts always using the same terms. Language use in domains, however, is filled with metaphors intermingled with formalized technical terms [3, 24]. Different members of an organization may refer to the same concept using different terms (synonymy), while different concepts may be represented by the same term (homonymy). Additionally, modelers, as initial designers of software systems, may intentionally deviate from the concepts identified in a domain and reconstruct them differently in a model to, e.g., better support maintainability of the system or integration with other systems [12]. Following this guideline, a case description may refer to a domain concept differently. For a hospital domain, e.g., the terms 'hospitalized person' and 'patient' may refer to the same concept and, consequently, should be modeled by a single class in a class diagram. Conversely, the expression 'book,' for example, may refer to a particular copy of a book, which may have a specific condition and sales price, or to a book in general of which many copies exist. To avoid conceptual redundancy and improve maintainability, a modeler should thus represent the concept 'book' through multiple classes.

Guideline 2 – ***Irrelevant domain information****: The description should contain domain information not required for the modeling problem at hand.* It can be reasonably expected that domains offer more information than is relevant for a specific model. To address a particular purpose, a model does not need to represent the domain in its entirety. Detecting those concepts and information relevant for a modeling purpose is thus a key competence students should acquire [9, p. 310]. As noted by Watanabe [23, p. 143]: "The importance of selecting what is useful out of an immense sea of worthless information can never be over-emphasized." This guideline can be addressed in case descriptions for modeling exercises by at least two means. First, a case description may include more information than is required for a specific modeling purpose. If that is the case, the modeling purpose must be clearly communicated within the exercise, for example by providing a set of analysis questions the solution model should focus on. Second, a case description may include more information than is relevant for a particular diagram type. A case description may include, e.g., detailed information about activities that should not be accounted for in class diagrams.

Guideline 3 – ***Construct ambiguity****: The description should avoid a clear mapping between domain information and modeling constructs.* Already in the 1980s, a similarity was pointed out between parts of speech (PoS) and constructs of software languages [1]. Chen, for example, notes that a "*common noun* (such as 'person,' 'chair') in English corresponds to an *entity type* in an ER diagram" [8, p. 130, italics in original]. Against this background, some researchers have developed approaches based on PoS tagging that support the automatic creation of UML models [e.g., 5]. However, such PoS–construct correspondences can serve as heuristics only and do not express strict rules that are uniformly followed, as also noted by Chen [8] himself. Manually crafted case descriptions may exhibit particularly clear PoS–construct correspondence, e.g., when a pre-developed solution model is translated ex-post into natural-language sentences. To address this guideline in case descriptions, two aspects can be considered. First, different PoS used in a case description may be mapped to the same construct. For example, in the sentence "*The system must capture at what time a flight has been booked*," no single noun like 'booking time' is used to express an attribute. Second, and more importantly, descriptions should confront students with key modeling decisions rather than providing unambiguous mappings to language constructs. For example, "*The age of each employee must be captured*," may suggest modeling `age` as an attribute to a class `Employee`. This, however, would threaten the integrity of the model since the particular age of an employee changes and must thus be regularly updated. A preferable solution for OOM would be to add an attribute `dateOfBirth` and return the current age through an operation.

Guideline 4 – ***Incomplete domain information****: The description should leave some aspects of a modeling solution unspecified.* Case descriptions should not explicitly state all information required for a modeling solution. Students should identify information gaps and fill them with reasonable assumptions. According to Moisan and Rigault [19, p. 45], this is a commonly encountered aspect in modeling practice. Incompleteness of provided domain information may concern (i) information necessary for creating a valid model, such as data types and multiplicities in a UML class diagram, or (ii) elements expected in a good modeling solution that are not explicitly mentioned. A common example may be the identification of reasonable generalizations in UML class diagrams when the case description does not explicitly mention candidate parent classes.

Guideline 5 – ***Abstraction level entanglement****: The description should include information referring to different levels of abstraction.* Object orientation supports seamless software development by providing constructs that can be used across the software development lifecycle [18]. Most common object-oriented languages are class-based, i.e., they require the specification of classes from which objects can be instantiated. While many software objects are assumed to represent perceivable empirical referents, modelers must abstract over these objects to formulate classes suited for

software development. As put by Nierstrasz [20, italics in original]: "[...] in the real world, there are only objects. *Classes exist only in our minds.*" This abstraction step should also be accounted for in case descriptions by entangling instance-level and type-level information. For example, a case description for a hospital may state: "*Last week, a patient called Paul Freeman came in with a high fever. Dr. Belle had a look at him and sent him home without any treatment plan.*" Faced with the task of creating a UML class diagram that can account for all provided information, students are forced to abstract over the provided information to induce/detect concepts such as 'patient,' 'reported symptoms,' 'administering physician,' and 'treatment plan.'

3 Example Application of L.I.C.I.A. Guidelines

We have formulated two variants of a case description following the L.I.C.I.A. guidelines. The example case description is described in Sec. 3.1. Each variant of the case description presented was provided as part of a modeling exercise for which students were asked to submit solutions. Our teaching context as well as the modeling tools used by students to complete these exercises is described in Sec. 3.2. Originally, we intended to conduct comprehensive multi-modal modeling studies [22] but after experiences with first participants, we decided to abort those mainly for two reasons. First, our original studies were limited to 30 minutes for completing the exercise, which, as it turned out, was far too short for students to provide a reflected modeling solution. Second, conducting studies with a baseline comparison of exercises using L.I.C.I.A. case descriptions and translation-oriented exercises were difficult because student solutions depend on the information provided in a case description. If one case description provides only example values for inducing data types (following guideline *A*) whereas another states the required data type more directly, the results are hardly comparable.

3.1 The Portaview Case Description

We applied the L.I.C.I.A. guidelines to formulate case descriptions used in modeling exercises for teaching software modeling with UML class diagrams. Two slightly different variants of a case description were developed for a fictional video-streaming platform called *Portaview*. The modeling exercises alongside reference solutions can be accessed in the original German or in an English translation at http://le4mm.org/portaview. In the following, an excerpt of variant B of the case description is presented and it is outlined how each guideline is addressed in the case description.

> *Denise has an account on Portaview. Portaview is a video platform where users can watch various kinds of videos. The videos are uploaded exclusively by users. Portaview only serves as a platform and does not provide any videos itself. Jestful71 uploaded a video named "Visiting Delaware" on 18 July 2024. Jestful71 is only his user name. His real name is Greg Sampson. [...] However, users must provide both names during registration but can choose to display only one of them.*
>
> *The video "Visiting Delaware" is a travel video. Music and sports videos are also available on Portaview. To watch videos, a user must subscribe to the account that uploaded the video. Denise has been subscribed to the channel of Jestful71 since 08 October 2023, for which she has been paying €5.99 monthly since then. Currently, each user is allowed to subscribe to only one other user. This restriction shall be lifted in the future. [...]*
>
> *Denise manages her favorite videos in her playlist "favorite videos." The playlist is private. This means that it is only visible to Denise. Users can also upload so-called "shorts," i.e., short videos. These can, in contrast to regular videos, also be looped. [...] However, shorts cannot be added to playlists.*

L – Label ambiguity. The descriptions refer ambiguously to the concept of an account, employing in the English translation to the terms 'user,' 'account,' and 'channel.' Within the descriptions, all three terms are closely intertwined and seem to share the same properties. At first, it is stated that users upload videos, but later, that videos are uploaded by accounts. In the second paragraph, it is first stated that "a user must subscribe to the *account* that uploaded the video," then "Denise has been subscribed to the *channel* of Jestful71" and, finally, that "each user is allowed to subscribe to only one other *user*." A single class is sufficient to account for the described case. Adding multiple classes would require clear conceptual distinctions that demand correctly adding information not provided in the case description.

I – Irrelevant domain information. Irrelevant information included in the description concerns, e.g., the platform Portaview itself. Since the solution model should describe the platform Portaview as a whole, no class like `Platform`, which would allow different particular platforms to be captured on the instance level, should be added to the solution. Likewise, the general information that users watch videos on Portaview does not need to be captured in a UML class diagram.

C – Construct ambiguity. The case description includes various examples of construct ambiguity. For example, to model subscriptions, two options may seem reasonable. First, a user class might have a `subscribes` association from and to itself. This is possible as long as users may subscribe to at most one other user, making the subscription to which the subscription-related attributes refer unambiguous. Second, subscriptions may be modeled as a separate class associated to a user class and including at least the subscription's start date attribute. Whether the monthly subscription fee is part of a user or a subscription class depends on whether it is assumed to be identical for all subscribers of a user. The second option is preferable since the case description states that the restriction on the number of subscriptions may be relieved in the future.

I – Incomplete domain information. Much information required for a valid model is not explicitly introduced in the case descriptions. This concerns, e.g., how many users may upload a single video or how many videos may be included in a playlist. The case description also introduces so-called "*shorts,*" which are implied to be a type of video. It is stated, e.g., that shorts "can, in contrast to regular videos, also be looped" and that they "cannot be added to playlists." Regular videos and shorts may also be assumed to share properties, such as being uploaded by a user. Modeling solutions are therefore expected

to introduce a generalization with an abstract class representing videos and subclasses for regular videos and shorts. Students must then decide which properties apply to both types (like genre or title) and which only to regular videos.

A – Abstraction level entanglement. The description intentionally varies between information about specific users and videos and information about how these concepts are interpreted within the context of Portaview. This is illustrated, for example, in the following part of the description: "*To watch a video, a user must subscribe to the account that uploaded the video. Denise has been subscribed to Jestful71 since 08 October 2023, paying a monthly fee of €5.99.*" Based on these sentences, students were expected to abstract from the specific subscription of Denise to identify attributes of subscriptions in general, including their start date and monthly fee.

3.2 Teaching Context and Used Modeling Tools

We teach an undergraduate course on enterprise modeling that includes a section on software modeling with UML class and object diagrams. The course is attended by students from a variety of study programs, among them Software Engineering and Business Administration. We decided to experiment with using the object-oriented multi-level modeling (MLM) language FMML^{x} (Flexible Multi-Level Modeling and Execution Language) [11] in our course because it integrates objects and class diagrams into one integrated model and not two separate artifacts that must be synchronized with each other. FMML^{x} models can be created with XModeler^{ML} [17]. FMML^{x} resembles standard UML notation. Just like UML, it provides users with the possibility to model classes with attributes, associations, and operations as well as objects with respective slots and links. FMML^{x} models can be executed and instantiated during runtime within XModeler^{ML}.

Students were asked to solve the Portaview modeling exercise with FMML^{x} and XModeler^{ML} in 2023. We reported our first experiences with using XModeler^{ML} in teaching UML class and object diagrams in [16]. Despite initial positive reactions from students, the tool was overwhelming for many and usability difficulties distracted them from the overall assignment. For this reason, we have developed a dialect of FMML^{x} called UML++ [16], which replicates the standard UML notation and restricts modeling on only two classification levels like standard UML. To support UML++ diagrams, we developed a new variant of XModeler^{ML}, called UML-MX [13, 14]. UML-MX allows users to model UML++ diagrams only and provides additional educational resources. UML-MX and XModeler^{ML} are both open source and can be downloaded at http://le4mm.org/xmodelerml. UML++ and UML-MX was used by students to solve the Portaview modeling exercise in 2024.

The use of these modeling tools has at least two important consequences that may affect our initial experiences with L.I.C.I.A. case descriptions. First, XModeler^{ML} and UML-MX provide students with the capacity to instantiate their created class diagram, analyze the instantiated objects, and update the class diagram again if deemed reasonable. Second, the modeling tool forces students to create valid UML class diagrams. It is not possible to leave data types or multiplicities unspecified. Some constructs of standard UML, which are also not discussed within our teaching context, are not supported by UML++, such as, e.g., association classes. The submissions of students using XModeler^{ML} and UML-MX were similar in quality. Thus, solutions created using XModeler^{ML} demand no special attention here. We assume only limited use of LLMs by students since LLMs struggle(d) to produce valid FMML^{x} models [15] and the use of LLMs for modeling was, as far as we perceived it, not widely distributed among students in 2023 and 2024.

4 Initial Observations of Student Solutions for Portaview Case Description

We received 29 student submissions created with XModeler^{ML} and 30 created with UML-MX, leading to a total of 59 unique student submissions for the Portaview case description. Submitting a solution was optional but students could receive bonus points for the exam based on the quality of the submission. Since L.I.C.I.A. case descriptions involve a higher degree of contingency they invite a broader scope of acceptable solutions, thus making the correction of students solutions more effortful. In the following, we report on how students handled specific aspects of the Portaview case description related to each L.I.C.I.A. guideline as detailed in Sec. 3.1. The underlying data, alongside a more in-depth overview, is provided at http://le4mm.org/portaview.

In the Portaview case description, the guideline *label ambiguity* refers especially to the modeling of the concept of user/account/channel. Approximately half of the submissions (47.5%) reconstructed the ambiguously referenced concepts differently from the reference solution, resulting almost always in modeling errors. Students frequently introduced separate classes for these concepts or overused generalization relationships to handle ambiguously referenced concepts, adding, e.g., an account class as a specialization of a concrete user class. In contrast, errors related to *irrelevant domain information* occurred less frequently. Only 8.5% of submissions added a class representing the platform Portaview, and 25.4% added irrelevant associations, most prominently a 'watches' association between users and videos.

Construct ambiguity produced the largest variety in the submitted solutions. Students differed in deciding whether domain concepts should be represented as classes, associations, or attributes. Generalization relationships were both overused by some students and underused by others. For example, many submissions accounted for short videos by adding a Boolean attribute to a video class instead of adding a separate subclass for short videos, preventing them from distinguishing between properties of regular videos and shorts. A considerable number of submissions (47.5%) also included errors in added generalization relationships, e.g., by not modeling a superclass as abstract or by modeling shorts as a subclass of regular videos, which made it possible to add them to playlists. Several submissions (47.5%) missed attributes that were indicated in the case description but not stated explicitly, e.g., an attribute to determine whether the real name or user name of a user is publicly displayed. Many students also struggled immensely with adequately modeling user subscriptions; 62.7% of submission included errors. Some solutions did not include any subscription association at all, while others introduced separate classes for users and subscriptions but added only one association between them.

Regarding *incomplete domain information*, students regularly completed missing information by making their own modeling

assumptions. For example, some submissions added attributes for playlist IDs or user registration dates, illustrating reasonable extensions beyond the information explicitly provided. At the same time, students did not always make adequate assumptions about the domain. A total of 47.5% of submissions specified association multiplicities more restrictively than intended, e.g., allowing playlists to include at most one video.

Finally, *abstraction level entanglement* was reflected in errors when abstracting from instance-level information. Inadequate data types for instance-level information occurred in 42.8% of submissions, while 33.9% missed attributes and 84.7% missed operations indicated by instance-level information. Moreover, 18.6% of submissions missed multiplicities implied by the provided instances.

These initial observations suggest the following. First, the modeling decisions introduced in the case description because of the L.I.C.I.A. guidelines are not straightforward for all students. For each modeling decision students were confronted with, varying proportions of students struggled to provide an adequate solution. Second, some decisions in modeling UML class diagrams seem particularly challenging to students. At least based on our teaching context, this includes deciding whether a class property should be modeled as an attribute or as an operation and when and how to add generalizations. While the frequency and intensity of observed modeling errors will likely vary in other teaching contexts, given differences in prior knowledge of students and teaching focus, our initial observations indicate that L.I.C.I.A. case descriptions can help uncover them.

5 Conclusion

In this paper we proposed a set of five guidelines for formulating case descriptions in modeling exercises abbreviated by the acronym L.I.C.I.A. Rather than presenting students with descriptions that already encode all intended abstractions, the guidelines encourage the inclusion of ambiguity, incomplete information, irrelevant details, and information on multiple abstraction levels that more closely resemble practical modeling situations.

To explore the educational implications of these guidelines, we developed case descriptions based on these guidelines and applied them in modeling exercises for UML class diagrams. Our initial experience with these modeling exercises indicates that students face challenges that extend beyond the correct application of UML notation, including resolving ambiguous domain concepts, identifying relevant information, and deriving abstractions from incomplete and instance-level descriptions.

Future work should investigate how such case descriptions can be integrated into different educational settings and extended to more advanced modeling topics, such as meta modeling and multi-level modeling. It may also be interesting to investigate how LLMs may be used to create conceptual models based on case descriptions following the guidelines proposed here. A more comprehensive evaluation and validation of the guidelines may be conducted, potentially informed by empirical studies on modeling practice. Apart from observable modeling practice, it is even more important to reflect upon modeling competencies students need to acquire for the future and design case descriptions that may better help convey them.

References

[1] Russel J. Abbott. 1983. Program Design by Informal English Descriptions. *Commun. ACM* 26, 11 (1983), 862–894. doi:10.1145/182.358441

[2] Mikael Berndtsson. 2005. Analyzing Course Configurations for Teaching Object-Oriented Modeling and Design. *IEEE Transactions on Education* 48, 2 (2005), 337–339. doi:10.1109/TE.2004.842891

[3] Richard J. Boland and Ralph H. Greenberg. 1992. Method and Metaphor in Organizational Analysis. *Accounting, Management, and Information Technology* 2, 2 (1992), 117–141. doi:10.1016/0959-8022(92)90004-C

[4] Torsten Brinda. 2003. Student Experiments in Object-Oriented Modeling. In *IFIP TC3 / WG3.2 Conference on Informatics Curricula, Teaching Methods and Best Practice (ICTEM 2002)*, Lillian Cassel and Ricardo A. Reis (Eds.). Springer, Berlin and Heidelberg, 13–20. doi:10.1007/978-0-387-35619-8_2

[5] Lola Burgueño, Robert Clarisó, Sébastien Gérard, Shuai Li, and Jordi Cabot. 2021. An NLP-Based Architecture for the Autocompletion of Partial Domain Models. In *33rd International CAiSE Conference*. 91–106. doi:10.1007/978-3-030-79382-1_6

[6] Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. 2023. On the Assessment of Generative AI in Modeling Tasks: An Experience Report with ChatGPT and UML. *Software and Systems Modeling* 22, 3 (2023), 781–793. doi:10.1007/s10270-023-01105-5

[7] Javier Cámara, Javier Troya, Julio Montes-Torres, and Francisco J. Jaime. 2024. Generative AI in the Software Modeling Classroom: An Experience Report with ChatGPT and Unified Modeling Language. *IEEE Software* 41, 6 (2024), 73–81. doi:10.1109/MS.2024.3385309

[8] Peter P. Chen. 1983. English Sentence Structure and Entity-Relationship Diagrams. *Information Sciences* 29 (1983), 127–149. doi:10.1016/0020-0255(83)90014-2

[9] Gregor Engels, Jan Hendrick Hausmann, Marc Lohmann, and Stefan Sauer. 2006. Teaching UML Is Teaching Software Engineering Is Teaching Abstraction. In *Satellite Events at the MoDELS 2005 Conference*. 306–319. doi:10.1007/11663430_32

[10] Hans-Georg Fill, Peter Fettke, and Julius Köpke. 2023. Conceptual Modeling and Large Language Models: Impressions From First Experiments With ChatGPT. *Enterprise Modelling and Information Systems Architectures* 18, 3 (2023), 1–15. doi:10.18417/emisa.18.3

[11] Ulrich Frank. 2014. Multilevel Modeling: Toward a New Paradigm of Conceptual Modeling and Information Systems Design. *Business and Information Systems Engineering* 6, 6 (2014), 319–337. doi:10.1007/s12599-014-0350-4

[12] Ulrich Frank. 2021. Language, Change, and Possible Worlds: Philosophical Considerations of the Digital Transformation. In *Crisis and Critique: Philosophical Analysis of Current Events*, Anne Siegetsleitner, Andreas Oberprantacher, Marie-Luisa Frick, and Ulrich Metschl (Eds.). De Gruyter, Berlin and Boston, MA, 117–138. doi:10.1515/9783110702255-009

[13] Ulrich Frank and Pierre Maier. 2025. How an unintended Side Effect of a Research Project led to Boosting the Power of UML. doi:10.48550/arXiv.2505.09269

[14] Ulrich Frank and Pierre Maier. 2025. UML-MX: Boosting Power of Object-Oriented Modeling and Enriching User Experience. In *EDOC 2024 Workshops*. 276–286. doi:10.1007/978-3-031-79059-1_17

[15] Pierre Maier and Vicky Kadziolka. 2026. Model Deepening with Large Language Models: Insights from Exploratory Studies with ChatGPT. In *ER 2025 Workshops Proceedings*. Springer, Cham, 119–136. doi:10.1007/978-3-032-08620-4_8

[16] Pierre Maier and Tobias Schwarz. 2024. UML++: Enhancing Student Learning of Object-Oriented Modeling through Executable Objects. In *MODELS Companion Proceedings '24*. 107–114. doi:10.1145/3652620.3687777

[17] Pierre Maier and Daniel Töpel. 2025. XModelerML v3: Integrating Executable UML with a Multi-Level Language Engineering, Modeling, and Execution Environment. In *MODELS Companion Proceedings '25*. doi:10.1109/MODELS-C68889.2025.00024

[18] Bertrand Meyer. 1997. *Object-Oriented Software Construction* (2 ed.). Prentice-Hall, Upper Saddle River, NJ.

[19] Sabine Moisan and Jean-Paul Rigault. 2010. Teaching Object-Oriented Modeling and UML to Various Audiences. In *Workshops and Symposia at MODELS 2009*. 40–54. doi:10.1007/978-3-642-12261-3_5

[20] Oscar Nierstrasz. 2010. Ten Things I Hate About Object-Oriented Programming. *Journal of Object Technology* 9, 5 (2010). doi:10.5381/jot.2010.9.5.e1

[21] Kristina Rosenthal, Stefan Strecker, and Monique Snoeck. 2023. Modeling Difficulties in Creating Conceptual Data Models. *Software and Systems Modeling* 22 (2023), 1005–1030. doi:10.1007/s10270-022-01051-8

[22] Kristina Rosenthal, Benjamin Ternes, and Stefan Strecker. 2020. Understanding Individual Processes of Conceptual Modeling: A Multi-Modal Observation and Data Generation Approach. In *Modellierung 2020*. Gesellschaft für Informatik e.V., Bonn, 77–92.

[23] Satoshi Watanabe. 1972. Information. In *Scientific Thought: Some Underlying Concepts, Methods, and Procedures*. Mouton and Unesco, Paris and The Hague, 129–144.

[24] Terry Winograd. 1986. A Language/Action Perspective on the Design of Cooperative Work. In *CSCW '86: Proceedings of the 1986 ACM Conference on Computer-supported Cooperative Work*. 203–220. doi:10.1145/637069.637096