A recommender system to assist conceptual modeling with UML

Conference Paper · June 2021

DOI: 10.18293/SEKE2021-039

Maxime Savary-Leblanc
University of Lille Nord de France
7 PUBLICATIONS **18** CITATIONS
SEE PROFILE

Xavier Le Pallec
Université de Lille
71 PUBLICATIONS **243** CITATIONS
SEE PROFILE

Sébastien Gérard
Atomic Energy and Alternative Energies Commission
243 PUBLICATIONS **2,901** CITATIONS
SEE PROFILE

A recommender system to assist conceptual modeling with UML

Maxime Savary-Leblanc*,** Xavier Le Pallec* Sébastien Gérard**
*Univ. Lille, UMR 9189 CRIStAL, F-59000 Lille, France **CEA LIST, Palaiseau, France
E-mail: maxime.savary-leblanc@univ-lille.fr

Abstract

This paper explores the understudied field of conceptual modeling assistance. *More specifically, we focused on the design and application of recommender systems as software assistants for conceptual modeling. Prior work on such systems has shown that trust plays a key role in the acceptance and exploitation of such systems. Consequently, as a starting point of our research, we applied established methods for constructing* multi-criteria recommender systems *(MCRS) to conceptual modeling in a way which could foster the emergence of trust. Finally, we chose supervised-learning techniques to refine and customize the recommendations generated by these systems. To help us determine the feasibility and practicality of our approach, we designed and implemented a prototype system that assists conceptual modeling with UML. Our system currently recommends class attributes when constructing UML class diagrams. A preliminary evaluation of this tool indicated a strong match between the recommendations provided by our system and personal choices made by the participants.*

1 Introduction

Domain knowledge is a prerequisite to produce software design and implementation tailored to stakeholders' requirements. One common way to formalize that knowledge is achieved through conceptual models, which are commonly used to describe or simulate a system. Acquiring such expertise requires to discuss with knowledgeable stakeholders and/or to get an access to useful documents, which both might not always be easily accessible.

In the same time, more and more model samples can be gathered from multiple sources, what represents an increasing number of already formalized and accessible knowledge pieces. For example, some companies keep archives of internal model repositories [1]. There also exist numerous open source projects that contain models [2] while some modeling tools even offer the possibility to create public projects that are free to browse. However, when it comes to hundreds of thousands models, the time spent browsing them manually and filtering the relevant ones seems unreasonable given the original problem.

DOI reference number: 10.18293/SEKE2021-039

One facility for exploiting this large amount of data is to build *recommender systems* [3] whose recommendations are based on such data. In our work, we decided to investigate the use of recommender systems to support the conceptual modeling activity. Our general research objective is to develop a comprehensive general methodology for designing recommender systems for conceptual modeling. As an initial step, we first developed a working prototype that can serve as a testbed to help in evaluating and refining our approach. The specific objective of this phase of our work was to develop a Multi-Criteria Recommender System (MCRS) that generates semantically meaningful attribute recommendations for UML classes. We used Roy's method to develop our approach and then performed an initial empirical evaluation as described in Section 5.

To the best of our knowledge, this is the first *semantic* recommender system for UML class modeling. As such, we feel that it is an important contribution to the modeling community. It provides a means of assisting the modeling process – something that could particularly benefit less experienced modelers. This can, in turn, lead to a broader adoption of model-based software engineering by software developers. In addition and in line with our overall objective, we believe that the approach applied here can be extended to cover other types of recommender systems to be used in modeling.

The remainder of the paper is organized as follows. In Section 2, we explain our approach, while in Section 3, we identify and formalize suitable recommendation criteria. A criteria aggregation method is described in Section 4. Section 5 presents the prototype tool and an initial experiment used to evaluate its suitability and effectiveness. Related work is covered in Section 6. Finally, Section 7 provides a brief summary followed by a discussion of potential future work.

2 Methodology

Most recommender systems implement one of the following three common recommendation methods: collaborative filtering [4], content-based techniques [5], or knowledge-based techniques [6]. Each has advantages and limitations. Common limitations include overspecialization, cold-start issues, and scalability [3]. To deal with these, different advanced recommendation approaches have been proposed, including multi-criteria recommender systems (MCRS) [7]. The quality of modeling recommendations invariably depends on more than one criterion. Thus, no single-criterion recommendation approach is likely to be adequate, which is why we chose to build our system as an MCRS. Such systems involve multiple criteria in selecting the final set of ranked recommendations. Adomavicius and Kwon [8] identify multiple selection techniques, such as rating-based, multi-objective optimization, or outranking relations. Because the user's goal cannot be inferred with certainty, and because extracting outranking relations between attributes seems semantically unreliable, we applied the rating-based selection technique. Thus, we use scores to determine the candidate attributes to recommend.

We followed Roy's methodology for analysing *multi-criteria decision-making* problems [9], as suggested by [8] to build our system. To do so, we first define the *object of decision* of our system, then a coherent *family of criteria* to assess candidate attributes, and finally a *utility function* which aggregates single criterion into a global score. The object of decision consists of all context elements that might be involved in deciding whether a candidate attribute represents an appropriate recommendation, as follows: attributes (i) selected by their nature or their presence in the hierarchy of classes or packages, (ii) represented in class diagrams, and (iii) tailored to the user of the modeling tool, who may have previous experience, personal preferences, and various levels of propensity to trust. All the candidate attributes can then be filtered and ranked based on the *utility function scores* (see Section 4), according to how well they perform on each criteria defined in Section 3.

3 Criteria formalisation

In this section, we describe the criteria and their rationale, on which each recommendation is assessed.

3.1 In-class recurrence criterion (C1)

In the following, the term 'owner class' of an attribute refers to the class that directly owns that attribute.

Rationale. The attribute is often present in classes with the same name as the owner class.

Selection filter. Attributes owned by classes with the same name as RC.

Rating approach. The most frequently occurring candidate gets the highest score and the least frequently occurring candidate gets the lowest non-null score. Candidates that do not appear in a class with the same name as the owner class get a null score.

3.2 In-class exclusivity criterion (C2)

Rationale. The attribute is only present in classes with the same name as the owner class.

Selection filter. Attributes owned by classes with the same name as RC.

Rating approach. Candidates which only appear in classes with the same name as the owner class get the highest score. Those that appear equally in all classes of the data set get the lowest score. Candidates that never appear in a class with the same name as the owner class get a null score.

3.3 Attribute synergy criterion (C3)

Rationale. The attribute is often present along with other attributes of the owner class.

Selection filter. Attributes connected to attributes owned by RC through their presence in a common class. Common classes are those classes that share the same name.

Rating approach. The more often a candidate and an attribute of RC appear together in a class, the higher the score. The more that a candidate appears together with different attributes of RC, the higher the score. Candidates which never appear together in a class get a null score.

3.4 Context similarity criterion (C4)

Rationale. The attribute often describes a class named the same as the owner class in similar models.

Selection filter. Attributes owned by classes named $|RC|$ in models which share at least two common classes with $M(RC)$.

Rating approach. Candidates from models that share the highest number of classes with $M(RC)$ get the highest score. Candidates from models which have no class in common get a null score.

4 Utility Function

The third step of our approach consists in constructing a utility function that aggregates the score of each single criterion into a global score, on which to base the ranking. In defining what makes a good explanation in recommender systems, [10] argue that *"justifying [a] recommendation is just half of the solution, the second half is to make it scrutable"*. To that end, in this section we first select an aggregation method that enhances system transparency. Then we emphasize support for context adaptability, and, finally, propose a determination process that allows system control through scrutability.

4.1 Utility Function selection

Adomavicius and Kwon [8] identify two major techniques for dealing with multi-criteria ratings to produce an overall rating: heuristic-based and model-based techniques. Heuristic-based techniques compute the score of each item for a given user, based on data derived from observing one specific user, using some heuristic assumption. To perform matching operations, these techniques often require specific knowledge about multiple users, based on their profile and from collaborative filtering. In contrast, model-based techniques generate a predictive model, typically using statistical or machine-learning methods that best explain the observed data. Once the model becomes available, they use it to estimate the score of individual recommendations.

In our case, the lack of data about the profiles of all users rules out heuristic-based techniques. On the other hand, model-based techniques using machine-learning methods enable the system to learn directly from the user, resulting in finely-tuned data. Consequently, we take a machine-learning model-based approach to determine the overall utility function. Note that, for greater system transparency, the aggregation process must be explainable. Therefore, rather than relying exclusively on machine-learning processes, which are rarely fully explainable, we define the utility function as a weighted sum of criteria rating functions.

We define this function as follows:

$$\begin{aligned} &\text{Let } (a,b,c,d) \in [0;1]^4 \text{ where } a+b+c+d=1, \\ &overall_{RC} \quad : \quad |\mathcal{A}| \quad \rightarrow \quad [0;1] \\ &\qquad\qquad\quad\;\; |p| \quad \mapsto \quad a \times s_1 + b \times s_2 + c \times s_3 + d \times s_4 \\ &\text{with } s_1, s_2, s_3, s_4 \text{ the scores for criteria 1 to 4} \end{aligned} \tag{1}$$

The machine-learning process is used to determine the values of the weights a, b, c, and d.

4.2 Context adaptability

Adomavicius and Kwon [11] also note that the aggregation function can have different scopes: total (i.e., when a single aggregation function is learned based on the entire data set), user-based, or item-based (i.e., when a separate aggregation function is learned for each user or item).

In the context of recommending UML attributes for classes, we identify four different possible *Contexts* i.e. situations. The system will provide recommendations for the following:

- **Context 1:** A class owning no attributes and no other classes in the model.
- **Context 2:** A class owning one or more attributes and no other classes in the model.
- **Context 3:** A class owning no attributes in a model but containing one or more other classes.
- **Context 4:** A class owning one or more attributes in a model and also containing one or more other classes.

Each of the above contexts has access to different information so that not all of the criteria can be applied equally to all of them. For instance, the context similarity criterion C4 relies on the presence of other classes in the model and is, therefore, not applicable to contexts 1 and 2. Consequently, we define the overall utility function in context k $overall_{k,RC}$ as:

$$overall_{k,RC}(|p|) = a_k \times s_1 + b_k \times s_2 + c_k \times s_3 + d_k \times s_4 \tag{2}$$

This results in four different utility functions corresponding to the four different contexts. They are determined individually in the course of the machine-learning process.

4.3 Utility function determination

The quality of a recommender system depends primarily on its ability to propose items that the user is likely to choose rather than items the user is unlikely to choose. Therefore, a high-quality recommender system must fit user preferences. Our system offers the possibility to reflect these preferences by assigning values to the weights of the four overall utility functions. This can be done manually, but finding suitable values would likely lead to suboptimal results. Instead, we chose a machine-learning approach to automatically determine these weights.

We collect labelled data through a dedicated interface (presented in section 5.1, and in the web page) during a preference-elicitation phase. This interface first presents multiple situations (a class diagram with recommendation target class) one at a time. A list of unranked candidates —potential recommendations— is displayed for each situation. Using this interface, the user is asked to remove all attributes that do not fit semantically in the presented situation. Once this is completed, the user is then asked to create a ranked list of the top 10 best recommendations from the displayed elements. This task should be repeated for multiple situations in different contexts a sufficient number of times in order to collect enough information to determine the four utility functions. Once this data is collected, it is used to calculate a_k, b_k, c_k, d_k weights in such a way that they maximize the Mean Top Average Precision metric defined in Section 5.2.2.

5 Implementation and Evaluation

In this section we first describe the implementation of our solution followed by a description of the initial evaluation and its results. More details about the implementation and its behaviour are available online[1].

5.1 Implementation

Recommender system. Our implementation conforms to the standard three-tier architecture pattern[2]: the data tier, the application tier, and the presentation tier. The data tier consists of a Neo4j[3] server which holds the full models data set represented as a graph. The application tier is a Spring Boot[4] server exposed as an API which is responsible for computing

[1]https://hufamo.univ-lille.fr/modeling-assistant
[2]https://www.tandfonline.com/doi/abs/10.1080/10580539608906981
[3]https://www.neo4j.com
[4]https://spring.io/projects/spring-boot

the scores and the output of the utility function, to produce recommendations. Finally, the presentation tier is a Papyrus[5] plugin which presents recommendations to the user and make it exploitable.

Supervised Learning Platform. The aim of the supervised learning approach is to determine a combination of a, b, c, d values that maximizes the accuracy of the system for a specific context. Initially, we start the learning process with equal values for these weights. These values are then varied using a predefined increment while maintaining the constraint specified by equation 1. For each configuration of a, b, c, and d, we calculate the chosen evaluation metric for the system and compare it to the previous maximum value. The highest value is stored as well as the associated configuration of weights. After all configurations are analysed, the one that maximized the chosen metric is selected. This algorithm is coded in Java as part of the application tier. To compute accuracy, the algorithm exploits the labelled data created through our dedicated interface. The **labelling interface** is a web application coded in HTML/CSS/JS that enables labelled data collection. It takes JSON files and class diagram pictures as input and outputs JSON files containing user preferences as output.

5.2 The evaluation

The preliminary evaluation of our approach is based of assessing improvements in the quality of the recommendations, as well as the adequacy of system control, information transparency, and system transparency. To the best of our knowledge, no similar approach can be found in the literature. A replication package contained the labelled data, the original files and the metrics source code is available online[6].

5.2.1 Data gathering

The evaluation involved data from over 95'000 models. These contained approximately 634'000 classes and 616'000 attributes. The models were retrieved from the GenMyModel[7] public repositories by courtesy of Axellience. For quality purposes, we only selected models greater than a minimum size (over 10 kilobytes).

We gathered labelled data according to the method described in Section 4.3. From this, we obtained 9,858 labelled attributes from 30 participants: 9 senior and 4 junior researchers, 3 senior and 12 junior developers from industry, and 2 M.Sc. students. Prior to starting the labelling exercise, participants were asked to answer questions about their familiarity with UML and the extent of their modeling work. On average, participants estimated their knowledge of UML class diagrams to range between fair and good (mean: 3.5 on 5-point Likert scale, std. deviation: 1.0). This assured us that participants had a relatively good understanding of the context and consequently, that the information gathered was semantically meaningful. Participants were asked to respond to up to 20 examples of different situations: 5 per context. In order to minimize the impact of participant fatigue on the results, the 20 examples were randomly displayed and participants were allowed to respond in several sessions.

[5]https://www.eclipse.org/papyrus/

[6]https://hufamo.univ-lille.fr/modeling-assistant

[7]https://www.genmymodel.com

5.2.2 Evaluation metrics

To more accurately evaluate the attribute recommendations, we compared the ranked results of our system with the ranked preferences as chosen by the users who created their top-5 ranked list. Consequently, we computed metrics for just the top-5 recommended attributes; i.e., the five attributes with the highest scores.

Precision@5 (P@5) is the proportion of recommended items that a user deemed as belonging in the top-5 list of relevant attributes. In our case, relevant attributes were those that were not excluded by the user from the candidate list.

$$P@5(\text{set}) = \frac{\text{n° of relevant items in system top-5}}{5} \tag{3}$$

TopPrecision@5 is the proportion of recommended items in the top-5 list provided by the recommender system that are also included in the top-5 set chosen by the user.

$$TP@5(\text{set}) = \frac{\text{n° of common items in user and system top-5}}{5} \tag{4}$$

TopAveragePrecision@5 (TAP@5) takes ranking into consideration in evaluating the mean average precision of the top-5 of the system. Mean Average Precision (MAP) is a popular metric for measuring recommendation algorithms in information retrieval. We defined TAP@5 as follows:

$$TAP@5(\text{set}) = \sum_{n=1}^{5} \frac{P(n) \times pos(n)}{R} \tag{5}$$

where pos(k) indicates whether the element from system top-5 in position k *matches* the position of the element in a user's top-5 list, while R refers to the number of elements for which $pos(k) = 1$; P(k) is the ratio of correctly recommended elements over top-k recommended elements.

These metrics can be computed for each ranked set of attributes. Therefore, as users provided several sets of attributes, we considered the means of these metrics as follows:

$$Mm(S) = \sum_{s \in S} \frac{m(s)}{N} \tag{6}$$

where m is the metric for which the mean is calculated (i.e., MP, MTP, and MTAP); S is the data set for which the mean was computed, and N is the number of elements in S.

Table 1. Labelled data distribution

	General	Ctx. 1	Ctx. 2	Ctx. 3	Ctx. 4
Training	8,146 (205 sets)	2,462 (50 sets)	2,338 (43 sets)	1,765 (59 sets)	1,581 (53 sets)
Testing	1,712 (40 sets)	544 (11 sets)	562 (10 sets)	303 (9 sets)	303 (10 sets)
Total	9,858 (245 sets)	3,009 (61 sets)	2,900 (53 sets)	2,068 (68 sets)	1,884 (63 sets)

Table 2. Learned overall functions

Context	Overall Function
Context 1	$0.80 \times s1 + 0.20 \times s2$
Context 2	$0.03 \times s1 + 0.01 \times s2 + 0.96 \times s3$
Context 3	$0.51 \times s1 + 0.03 \times s2 + 0.46 \times s4$
Context 4	$0.56 \times s1 + 0.03 \times s2 + 0.29 \times s3 + 0.12 \times s4$

5.2.3 Metrics results

We obtained 245 sets of labelled data from users, which constitute a corpus of 9,858 attributes distributed for training and testing phases, as presented in Table 1. We trained our overall rating functions with 81% (205 sets) of the total labelled data set and obtained the functions presented in Table 2. The goal of the training was set to the maximization of the MTP@5 metric, as it is the most representative possible improvement of our system when compared to unassisted user selections.

The added value of using machine-learning is demonstrated by the evolution of the metrics before and after the machine-learning process. Both situations only differ in the the values and distribution of weights in the utility functions. We set up the initial configuration (i.e. before ML) by setting the weight values to be equal. For instance, we define the initial aggregation function for Context 4 as $overall_4 = 0.25 \times s1 + 0.25 \times s2 + 0.25 \times s3 + 0.25 \times s4$. The final configuration corresponds to the application of the utility functions defined in Table 2.

We evaluated the overall utility functions on a specific testing data set, which represents 19% of the full labelled data set, (see Table 1). The other 81% were used for training purposes. The results of the metrics evaluation are presented in Table 3.

5.3 Discussion

The initial general MP@5 is pretty high (87.0%). The low impact of the learning process on this score (+4.5%) indicates that the different criteria already strongly converge to recommend user-relevant attributes, and that the impact of the weights on the overall rating functions are, in that case, secondary. The impact of the supervised-learning process becomes more important according to the desired quality of the recommendations. Indeed, the initial low value of MTP@5 increases from 34.7% to 51.0% after the learning process. This means that, on average, more than two attributes of the 5 first recommendations of the system are attributes that participants included in their top-5 best recommendations. Following the learning step, MTP@5 shows the most significant increase among all metrics (+16.3%). The utility functions were defied so as to optimize this metric. MTAP@5 takes differences in recommendations ranking between system and user top-5 into account. Only high-quality recommendations increase this metric, which explains why it has the lowest initial values for all contexts. With a final value of 42.5%, MTAP@5 indicates that, on average, more than 2 attributes of the 5 first recommendations are in participants' top-5, likely to be in top positions and ordered as the participants expected.

5.4 Evaluation results

The empirical results obtained indicate that our approach provides acceptable results (on average, more than 4 recommended attributes out of 5 are deemed relevant, and also 2 recommendations out of 5 appear in users' top-5 rankings). However, as pointed out earlier, it is too early to make any firm conclusions about the effectiveness of our approach compared to alternatives until further evaluations are performed. In addition, we can draw the following conclusions from the evaluation:

- the initial effectiveness measure that we proposed here looks as if it could serve as a common metric for future related work.
- The defined criteria do seem to reflect information trustworthiness. Moreover, the rationale behind them can be easily explained, which means that they do support information transparency.
- The linear utility function approach we used allows any overall score to be traced to each criterion used to derive it. This enables users to understand the inner mechanisms of the system and thus supports system transparency.
- The utility function can either be set manually or defined using supervised-learning. These settings allow users to have control over the results that are presented giving them control of the system.

6 Related work

In this section, we review published work in the following related areas: (i) tools that help with semantics-related issues involved in modeling, and (ii) recommender systems for software engineering.

Table 3. Testing data set metrics measures

Metric	General	Ctx. 1	Ctx. 2	Ctx. 3	Ctx. 4
Initial MP@5	87.0%	83.6%	90.0%	84.4%	90.0%
Initial MTP@5	34.7%	23.6%	38.7%	36.7%	40.0%
Initial MTAP@5	27.8%	4.5%	22.5%	11.1%	35.0%
Final MP@5	91.5% (+4.5%)	92.7% (+9.1%)	90.0% (-)	93.3% (+8.9%)	90.0% (-)
Final MTP@5	51.0% (+16.3%)	56.4% (+32.7%)	49.3% (+10.7%)	52.2% (+15.5%)	46.0% (+6.0%)
Final MTAP@5	42.5% (+14.7%)	50.0% (+45.5%)	35.0% (+12.5%)	50.8% (+39.7%)	45.0% (+10.0%)

6.1 Conceptual modeling assistance

Although a lot of work has been done on supporting software engineering with software assistants, not much of it has been applied to modeling. Segura et al. [12] recognize the need for assistance during modeling activities and introduce Extremo, an Eclipse plugin for modeling. They propose a framework for integrating diverse data sources into the Eclipse modeling environment. However, the data sources, such as model repositories, must be provided by the user. Koschmider et al. [13] propose a recommendation-based editor for business process modeling. Their system provides users with recommendations about partial process models.

Kogel [14] describes the early stages of a work on modeling recommendations and proposes a prototype providing unranked recommendations. Elkamel et al. [15] present a UML class recommender system that recommends new classes for a UML model. This system measures the similarity between current model classes and existing ones from a repository to recommend the closest matches. In a similar fashion, Cerqueira et al. [16] proposed a content-based approach for recommending UML sequence diagrams.

The above papers highlight both the novelty of and the need for semantic assistance in modeling activities. Surprisingly, while class diagrams remain among the most widely exploited UML diagrams [17], almost no effort has been conducted to address support for their design.

6.2 Recommender systems for software engineering

Only a few recommendation systems have been applied to modeling to date. In fact, Dyke et al. [18] identify recommenders for *modeling* as a promising new area of research, since recommender systems have already found their way into general software engineering [19] and many of the software lifecycle processes, as defined in ISO/IEC 12207 [20].

Multiple works [21] [22] investigated the application of recommender systems to the field of requirements engineering. Sharma and Sodhi proposed a recommender system [22] to help in dealing with the manual effort required to identify and analyse relevant architectural patterns in the context of a particular set of software requirements. A variety of recommender systems focus on the software construction process, from providing code examples to suggesting modifications [19]. The work described in [23] provide developers with recommendations about API usages and parameters. In [24], Allamanis et al. propose an algorithm which suggests meaningful class and method names to enhance software quality. Some works also involve recommenders for finding relevant answers to developer's technical questions [25].

All of the above clearly identifies a gap in support for recommender systems in semantic-based assistance for conceptual modeling.

7 Conclusion and Future Work

In this paper, we are seeking to support the conceptual modeling task. As an initial step we proposed, implemented, and evaluated a modeling recommender system. The initial evaluation, involving both practitioners and students and a prototype implemented with Papyrus, indicates that the approach holds promise. A replication package was provided to serve as first comparison point for future works in the domain.

Our plan is to generalize the approach realized in the prototype for other types of models, such as activity and sequence diagrams, to move towards our greater objective of a generic framework for building design-assisting recommender systems. We also plan to conduct further work specifically focusing human-centric aspects. This includes ways to provide users with explanations about recommendations, but also the types of interaction recommender systems should propose to best fit users' mental design process.

References

[1] Z. Yan, R. Dijkman, and P. Grefen, "Business process model repositories – framework and survey," *Information and Software Technology*, vol. 54, no. 4, pp. 380 – 395, 2012.

[2] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use uml: Mining github," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 173–183.

[3] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender system application developments: A survey," *Decision Support Systems*, vol. 74, pp. 12–32, Jun. 2015.

[4] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, *Collaborative Filtering Recommender Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 291–324.

[5] M. J. Pazzani and D. Billsus, *Content-Based Recommendation Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341.

[6] R. Burke, "Knowledge-based recommender systems," in *ENCYCLOPEDIA OF LIBRARY AND INFORMATION SYSTEMS*. Marcel Dekker, 2000, p. 2000.

[7] N. Manouselis and C. Costopoulou, "Analysis and Classification of Multi-Criteria Recommender Systems," *World Wide Web*, vol. 10, no. 4, pp. 415–441, Dec. 2007.

[8] G. Adomavicius and Y. Kwon, "Multi-Criteria Recommender Systems," in *Recommender Systems Handbook*,

F. Ricci, L. Rokach, and B. Shapira, Eds. Boston, MA: Springer US, 2015, pp. 847–880.

[9] B. Roy, *Multicriteria Methodology for Decision Aiding*. Springer Science & Business Media, Nov. 2013, google-Books-ID: lf7lBwAAQBAJ.

[10] N. Tintarev and J. Masthoff, “A survey of explanations in recommender systems,” in *2007 IEEE 23rd International Conference on Data Engineering Workshop*, 2007, pp. 801–810.

[11] G. Adomavicius and Y. Kwon, “New recommendation techniques for multicriteria rating systems,” *IEEE Intelligent Systems*, vol. 22, no. 3, pp. 48–55, 2007.

[12] Ángel Mora Segura and J. de Lara, “Extremo: An eclipse plugin for modelling and meta-modelling assistance,” *Science of Computer Programming*, vol. 180, pp. 71 – 80, 2019.

[13] A. Koschmider, T. Hornung, and A. Oberweis, “Recommendation-based editor for business process modeling,” *Data & Knowledge Engineering*, vol. 70, no. 6, pp. 483 – 503, 2011.

[14] S. Kögel, “Recommender system for model driven software development,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York: Association for Computing Machinery, p. 1026–1029.

[15] A. Elkamel, M. Gzara, and H. Ben-Abdallah, “An uml class recommender system for software design,” in *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, 2016, pp. 1–8.

[16] T. Cerqueira, L. Marinho, and F. Ramalho, “A Content-Based Approach for Recommending UML Sequence Diagrams,” Jul. 2016.

[17] D. Akdur, V. Garousi, and O. Demirörs, “A survey on modeling and model-driven engineering practices in the embedded software industry,” *Journal of Systems Architecture*, vol. 91, pp. 62–82, Nov. 2018.

[18] A. Dyck, A. Ganser, and H. Lichter, “On designing recommenders for graphical domain modeling environments,” in *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2014, pp. 291–299.

[19] M. Robillard, R. Walker, and T. Zimmermann, “Recommendation systems for software engineering,” *IEEE Softw.*, vol. 27, no. 4, p. 80–86, Jul. 2010.

[20] “Iso/iec/ieee international standard - systems and software engineering – software life cycle processes,” *IEEE STD 12207-2008*, pp. 1–138, 2008.

[21] N. Hariri, C. Castro-Herrera, J. Cleland-Huang, and B. Mobasher, *Recommendation Systems in Requirements Discovery*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 455–476.

[22] S. Sharma and B. Sodhi, “Apr: Architectural pattern recommender,” in *Proceedings of the Symposium on Applied Computing*, ser. SAC ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1225–1230.

[23] F. Thung, S. Wang, D. Lo, and J. Lawall, “Automatic recommendation of api methods from feature requests,” in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013, pp. 290–300.

[24] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, “Suggesting accurate method and class names,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 38–49.

[25] T. Du, J. Cao, Q. Wu, W. Li, B. Shen, and Y. Chen, “Cocoqa: Question answering for coding conventions over knowledge graphs,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 1086–1089.