Solving a Multi-criteria Decision Problem to Efficiently Handle UML Model Inconsistencies

Driss Allaki
Institut National des Postes et Télécommunications, Rabat, Morocco.

*Corresponding author. Tel.: +212 673 331 551; email: d.allaki@inpt.ac.ma
Manuscript submitted September 15, 2022; revised November 20, 2022; accepted January 25, 2023.
doi: 10.17706/jsw.18.2.70-85

Abstract: Nowadays, every business is heavily depending on software. However, designing and developing software is a very serious engineering challenge since software systems are growing in size and complexity. This gives rise to many inconsistencies in software design. To deal with this issue, researchers have been working for many years on different model inconsistency management activities, namely the detection, the diagnosis and the handling of those inconsistencies. This work focuses on handling UML model inconsistencies by proposing an AHP-based method aiming to help modelers choosing the right repair action that fits well their modeling objectives and strategies. The proposed method helps structuring the selection of the most appropriate repair action among a set of alternatives. It also represents and quantifies the criteria elements of this decision problem, relates these elements to overall goals and evaluates and ranks alternative solutions. The efficiency of the method is evaluated using different inconsistency examples through different situations. The obtained results show that modelers can have credible propositions to make decisions in accordance with their initial objectives.

Key words: UML models, handling inconsistencies, decision problems, analytic hierarchy process (AHP).

1. Introduction

From a very early age, making life easier was and is still one of the primary goals of human kind. For this end, every practical act, in almost all the aspects of the life, is essentially investigated to feel a need in less time and effort and with more ease and reliability. The idea of making these intentions into practice seems to be an obvious task. However, in the majority of the time, it needs to be rigorously thought and designed.

Thus, engineers, scientists and practitioners build complex structures and systems using expressive graphical representations such as models. In some cases, these models are physical, like mock-ups of houses or airplanes. In other cases, the models are less tangible, as seen in business financial models or electrical circuit diagrams. In all of these cases, a model serves as an abstraction or an approximate representation of the real item that is being built [1]. From the same perspective, it is technically and economically wise in software development to build complex and critical systems by first creating a model of the software being developed.

Such model provides the ability to visualize the entire system, understand it and assess different options before taking on the technical and financial risks of that system. It is also a major tool for communication among the different stakeholders in a project. Every member of the team, from users to developers, uses and enriches models in different ways. Furthermore, the model has the particular advantage of facilitating system traceability, i.e. the ability to start from one of its components and monitor its interactions and relationships with other parts of the model.

Being aware of the importance of models, the software industry has adopted the Unified Modeling Language (UML) [2] as a de facto standard for describing software models and related artifacts. UML contains a set of related diagrams that specify, visualize, construct and document all the aspects of a software system. In turn, a software modeling process defines a sequence of steps, partially ordered, which contribute to the realization of software or changing an existing system [3]. According to Huzar et al. [4], the incremental and iterative nature of software modeling processes, the multi-view and sometimes distributed software systems in addition to the overlapping and synchronous nature among different modeling artifacts in UML can give rise to several inconsistencies.

An inconsistency roughly means that overlapping elements of different model aspects do not match each other. Or in other words, it is a state in which one or more diagrams of UML model portray conflicting being syntactically or semantically in contradiction, incomplete, or reflecting ambiguities or anomalies [5].

These inconsistencies can be the source of many errors and can therefore invalidate the models and complicate the whole software development process. In fact, many inconsistencies in models lead to many bugs and anomalies in software design. Because in a larger context, they can also enable MDA (Model-Driven Architecture) models [6] that contain sufficient information for automatic transformations (i.e. Platform Independent Models (PIMs)) and complicate the model transformation process. Therefore, the expected benefits of the MDA may be directly affected by the various effects of the problem. For example, if the software system information we want to retain permanently is represented in an incomplete, ambiguous, or contradictory model, the sustainability of the model expected by MDA will be of little value because the latter can easily lose its meaning and thus its capabilities. Furthermore, the productivity gains that MDA seeks and consideration of execution platforms may in turn be adversely affected by inconsistency. In fact, MDA integrates the models in the model transformation into the production process of the application. However, if these models do not contain correct and sufficient information for possible automatic code generation, they will not function properly. Additionally, many of the basic operations that apply to models, such as merging, matching, slicing and splitting are increasingly automated to achieve productivity gains at various stages of the application lifecycle. However, these operations may cause some problems such as duplication of some items or non/multiple matches of other items.

Experiments in [7] confirm these statements, which show that the sheer number of inconsistencies in UML design is considerable. And in industrial practice, the design is pushed to the implementation stage while there are still a lot of inconsistencies in the model [8].

Therefore, special attention should be paid to the management of inconsistencies in order to understand changes in the software life cycle, correct errors, adapt to new requirements, etc. In fact, inconsistency management is defined in [9] as "*the process of managing inconsistencies between software models to support the goals of relevant stakeholders*". More specifically, the inconsistency management process includes the activities of detecting (identifying), diagnosing, and handling (repairing / dealing with) inconsistencies.

Inconsistency detection is the activity of checking for inconsistencies in software models [10]. Various proposals (for instance [11-13] and [14]) for this have been made in the literature. A Systematic Literature Review about detecting inconsistencies was presented in [15-42].

Diagnosing inconsistencies involves "identifying the source, cause, and impact of an inconsistency" [15]. Furthermore, different types of inconsistencies can be distinguished. In fact, the classification of inconsistencies must be carefully defined to facilitate its diagnosis.

On the other hand, the activities of handling inconsistencies include generating and identifying possible actions to deal with them, choosing one of the actions to take, and evaluating and managing the risks and benefits that may arise when applying the chosen resolution action.

Over the past two decades, a lot of work has been proposed to address the inconsistency of UML models. Of these, some focus on detecting inconsistencies, while others focus on the best way to resolve them. Regardless of the motivation or the method used, the benefit sought by ensuring model consistency is to minimize the cost, time, and effort upstream of the software development process. This can be ensured by creating consistent models that provide reliable and error-free code; which can be an important step in a model-driven approach in software development.

From the above sub-activities of dealing with inconsistencies, we propose to provide a solution that helps the modeler choose the correct repair action based on the followed modeling strategy. This choice is justified by the lack of contributions in the field; as most proposals focus only on generating possible repair operations or managing dependencies between inconsistencies. In short, this paper answers the following question: *How can we help modelers choose appropriate resolution actions to fix identified inconsistencies?*

The remainder of this paper is structured as follows. Section 2 provides more background knowledge about the existing handling strategies and the different resolution actions. Section 3 explains the principle of the AHP-based method used to determine the prioritized repair action. Section 4 presents and discusses the obtained results of the carried-out experimentation. Section 5 compares the paper's proposal with related work. Section VI concludes the paper and announces future work directions.

2. Background: Handling Strategies and Repair Actions

Before introducing the work carried out in this paper, we briefly cite and present how to handle inconsistencies in UML models according to [10]. First, we present the different existing handling strategies. Then, we focus on repair actions and their related issues and challenges.

2.1. Handling Strategies

The activity of inconsistency handling is concerned with the questions of how to deal with inconsistencies? What are the impact and consequences of specific ways to deal with inconsistencies, and when to deal with inconsistencies?

In what follows we will focus on the ways of dealing with inconsistencies. In fact, different handling strategies exist and combinations of them can occur. For example, some researchers [16] tolerate inconsistencies by ignoring or deferring them. Others deal with them by differentiating between changing and non-changing actions [15].

1) *Non-changing actions:* only notify the user or perform analysis for safe further inference from inconsistent models.
2) *Changing actions:* can be divided into partial resolution actions and full resolution actions.
    - *Partial resolution actions:* improve the model for a single inconsistency, but not fully resolve the inconsistency of the model.
    - *Full resolution actions:* modify the *model* to completely resolve the inconsistency. Most of them are a combination of different partial repair actions.

Recall that the diagnostic activity involves sources, causes, and effects of inconsistencies. In fact, the source of the inconsistency is the subset of UML elements involved in the inconsistency. As mentioned earlier, the cause of the inconsistency can be important when dealing with inconsistencies. In addition, determining the impact of inconsistencies may be necessary to determine the priority of handling inconsistencies. However, the impact of inconsistency depends on the application and on the current activities of the software development life cycle. Therefore, determining these effects is a task that is usually delegated to the modeler.

2.2. Repair Actions

Inconsistency resolution is one of the main objectives considered in the inconsistency management process. However, doing this automatically is very difficult. In most cases, resolution of inconsistencies includes changes in the model that also change the semantics of the model. In this case, a lot of user interaction is required. In fact, the resolution of inconsistencies may require user interaction, automatic changes of the model or a combination of both. In all these cases, repair actions, also called resolution actions, may include a set of guidelines that are provided to the user to repair inconsistencies. An action can also be a predefined set of steps specifying a solution to a specific inconsistency.

There are three different types of resolution actions:

- *Add model element:* It concerns the action of creating a new model element. In terms of UML metamodels, this means *instantiation* of metaclasses.
- *Remove model element:* It *concerns* the action of removing model elements. This means deleting an instance of a specific metaclass.
- *Change model element:* It *concerns* the action of modifying a model element by changing one of its properties. Under the hood, this involves two actions, namely deleting and creating connections between UML model elements.

2.3. Challenge: Selecting the Appropriate Repair Action

Various techniques can be used to help users resolve inconsistencies. In fact, the choice of resolution depends on the nature and cause of the inconsistency. However, it is difficult to (automatically) detect the cause of this inconsistency without additional knowledge about previous model changes or the user. Therefore, in most cases, the user has to decide what action needs to be taken. However, even knowing previous model changes, sometimes only some remedies are more likely to be performed than others based on the cause of the inconsistency. This leads to considering the following challenge: "*How do we choose an appropriate repair operation to resolve the inconsistency?*"

In what follows, we will answer on how to deal with this challenge using our semi-automatic solution.

3. The Proposed Approach

The challenges we are addressing in this work are typical decision-making situations. In fact, many techniques such as AHP [17], ANP [18], ELECTRE [19], DEMATEL [20], PROMETHEE [21] and TOPSIS [22] are used to solve multi-criteria decision-making problems. Each of these techniques is used to deal with a specific class of problems. More details on these techniques and their applications are provided in [23].

In this paper, Analytic Hierarchy Process (AHP) is used to organize and analyze decision-making situations because it aligns with the challenges we are dealing with, in addition to its flexibility and ability to help find the most suitable original options for the goals of the modeler and vision.

In what follows, we present the general steps of the usual AHP technique (independently on the handling inconsistency context), and then we explain and illustrate our proposed AHP-based solution to help the modelers choose the most appropriate repair action.

3.1. The Analytic Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) [17] is a multi-criteria decision making technique based on the evaluation of a set of criteria and alternatives to reach a specific goal. AHP returns the most relevant alternative with respect to the set of the pre-selected criteria. As stated in [24], the AHP method is based on three main steps: *3.1.1.* model structure *3.1.2.* pairwise comparison (comparative judgment of the criteria and/or alternatives) and *3.1.3.* synthesis of the priorities.

3.1.1. Model structure

In AHP, the problem is structured in a hierarchy linking the goal and the different alternatives through a set of criteria.

3.1.2. Pairwise comparisons

Once the hierarchy is built, the method performs pairwise comparisons (a series of measurements) in order to compute the priorities of (a) each criterion with respect to its importance to reach the goal, and the priorities of (b) each alternative with respect to its strengths in meeting each criterion. The results of these comparisons will be entered into a matrix that is processed mathematically to derive the priorities for all the nodes on the same level (criteria or alternatives). It is worth mentioning that a consistency checking algorithm is executed in order to ensure the consistency of each matrix. The priorities will then be combined throughout the hierarchy to give an overall priority for each alternative. The alternative with the highest priority will be the most suitable, and the ratios of the alternatives' priorities will indicate their relative strengths with respect to the goal.

3.1.3. Synthesizing final priorities

Now that the priorities of the Criteria with respect to the Goal, and the priorities of the Alternatives with respect to the Criteria are known, the final and third step is to deduce the priorities of the Alternatives with respect to the Goal using some arithmetic calculations. These priorities are then ranked to propose the most suitable alternative.

3.2. Incorporating AHP Technique into Repair Action Determination

The AHP technique can be followed to lay the foundation of a framework that helps modelers answer the question: "*What is the appropriate repair action to be chosen in order to fix an inconsistency?*"

Fig. 1 presents the instantiation of the AHP technique in the context of choosing the most suitable repair action for a specific inconsistency. This instantiation is established with a set of activities that can be carried out to achieve the targeted goal in an inconsistency handling context.

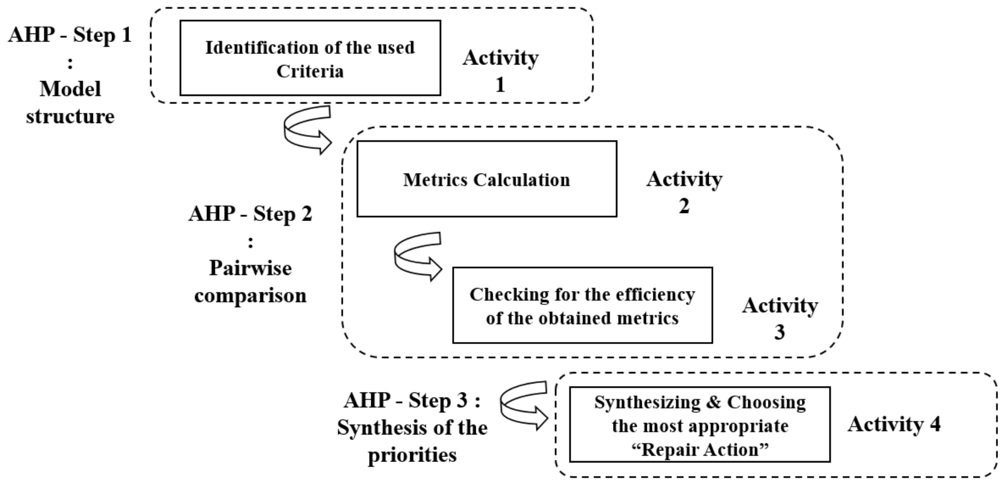

AHP - Step 1 : Model structure
Identification of the used Criteria
Activity 1
Metrics Calculation
Activity 2
AHP - Step 2 : Pairwise comparison
Checking for the efficiency of the obtained metrics
Activity 3
AHP - Step 3 : Synthesis of the priorities
Synthesizing & Choosing the most appropriate "Repair Action"
Activity 4

Fig. 1. Applying AHP to choose the most suitable repair action.

Before providing more details, we mention that the choice of a resolution operation is determined by the modeling phase or the strategy followed by the modeler. In fact, the actions taken to resolve inconsistencies may vary while the model is being built or is in the refactoring or recasting phase. For example, a modeler is more likely to add elements when building a model for the first time. However, when her/his vision is to rebuild or redesign the model, she/he will encourage removing elements from the model.

With this in mind, we consider the following activities to select the most appropriate repair operation:

Activity 1 — Identification of the used criteria: It is important to identify potential criteria that should be considered when applying the AHP technique in order to select remediation measures that address the inconsistency. In fact, these standards are set by the modeler. This user's involvement is motivated by the intention to produce more specific criteria for specific cases that only the modelers can better understand; which lead to more appropriate decisions.

The other point to consider is that these criteria have to be inclusive rather than exclusive. This means that the semantics of a criterion should not prevent the consideration of the others when calculating the priorities.

In addition to that, the determined criteria should be aptly named to facilitate the execution of the other activities of the AHP decision making process.

Activity 2 — Metrics calculation: Calculating metrics is an activity based on the sub-steps of the AHP pairwise comparison. The aim of this activity is to determine the different AHP priorities by comparing the criteria in accordance to the goal (choosing the most appropriate repair action) and by comparing the alternatives (the existing repair actions) to the criteria.

These comparisons between pairs of elements can come from different sources such as personal disposition of the decision makers, experts‟ pre-defined judgments or can be based on concrete data. In all these cases, the strength of AHP is that it represents them all in numerical priorities.

Activity 3 — Checking for the efficiency of the obtained metrics: The obtained metrics (i.e. priorities) in Activity 2 may be non-efficient (may lead to falsified results and decisions) since the elements of the hierarchy (i.e. criteria and alternatives) can be tangible or intangible, carefully measured or roughly estimated, well or poorly understood. Thus, it is mandatory to check the efficiency of these metrics by applying the consistency checking algorithm of the AHP technique. Recall that this algorithm is mathematically proved [26] and is about calculating a consistency ratio that has to be less than the 0.1 value.

Activity 4 — Synthesizing and choosing the most appropriate "Repair Action": Performing this activity is based on the AHP third step (i.e. synthesizing final priorities). It is a matter of calculating numerical priorities of the decision alternatives (the repair actions). These numbers represent the relative ability of the repair actions to handle the inconsistency. The final result gives a ranking of repair actions. The alternative having the higher score is the repair action proposed by the method to be executed. However, it is worth noticing that it is up to the modeler to take the final decision concerning the execution of the proposed repair action.

3.3. Proof of Concept and Illustration

To illustrate the functioning of the proposed AHP-based method, we chose to apply it on the "*dangling type reference*" inconsistency and present the different activities of the method and their related results as already shared in this initial work [39]. This inconsistency occurs when "a type of a parameter or an attribute refers to a class that does not exist in the model". In the majority of cases, this inconsistency occurs owing two main reasons. The type of the parameter involved may have been removed or the type of the attribute is not yet included in the model. Consequently, the list of Resolution Actions (RA) that are able to fix this inconsistency are enumerated as follows:

- (RA.1): *"Adding the type to the UML model involved."*
- (RA.2): *"Replacing the type of the attribute or parameter by an existing type in the model involved."*
- (RA.3): *"Removing the involved attribute or parameter."*

These listed repair actions are the three options among which we have to choose one.

3.3.1. Activity 1

In this example, we assume that, in addition to the development phase of the model, the criteria extraction is also inspired by the different strategies that modelers may follow when designing software systems. In our case, we make the assumption that the model is not yet completed and that it is still under construction.

Thus, the considered criteria (i.e. modeling strategies) are as follows:

1) *Radical Construction (RC):* reflects the purely constructive orientation of the modeler. It is usually given priority when modelers want to add elements to their models.
2) *Corrective Construction (CC):* represents the vision of changing elements while building the model.
3) *Radical Refactoring (RR):* reflects the pure refactoring orientation of the modeler. It is more likely to be favored when the overall vision is to change an element or its properties.
4) *Destructive Refactoring (DR):* means that the general vision is to encourage the removal of some model elements when attempting to refactor the model.
5) *Radical Destruction (RD):* expresses the extreme vision of removing elements from the model.

As a result, the AHP hierarchy for choosing the most suitable repair action to fix the "*dangling type reference*" inconsistency is illustrated in Fig. 2.

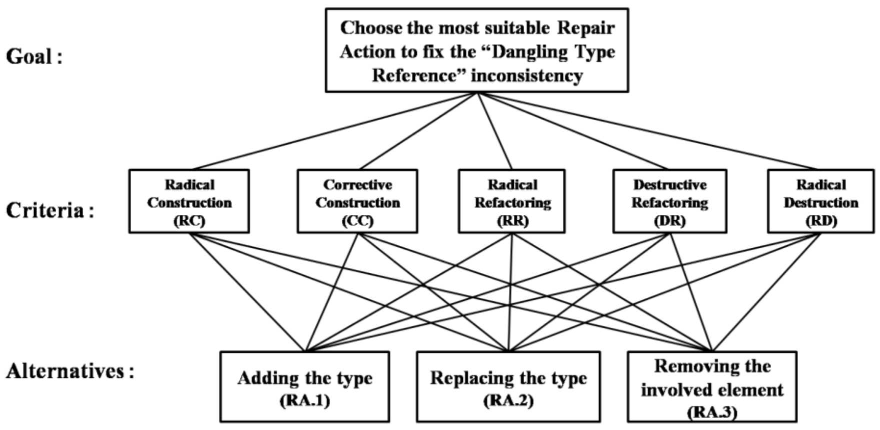

Goal :
Choose the most suitable Repair Action to fix the "Dangling Type Reference" inconsistency
Criteria :
Radical Construction (RC)
Corrective Construction (CC)
Radical Refactoring (RR)
Destructive Refactoring (DR)
Radical Destruction (RD)
Alternatives :
Adding the type (RA.1)
Replacing the type (RA.2)
Removing the involved element (RA.3)

Fig. 2. The AHP hierarchy for choosing the most suitable repair action.

3.3.2. Activity 2

After defining the AHP technique ingredients, namely the alternatives and criteria to be used, the next step is to calculate the required metrics by creating the criteria comparison matrix.

In fact, using the rankings defined in [25], we assume that each criterion is correlated with the others as shown in Table 1 (Criteria vs the Goal).

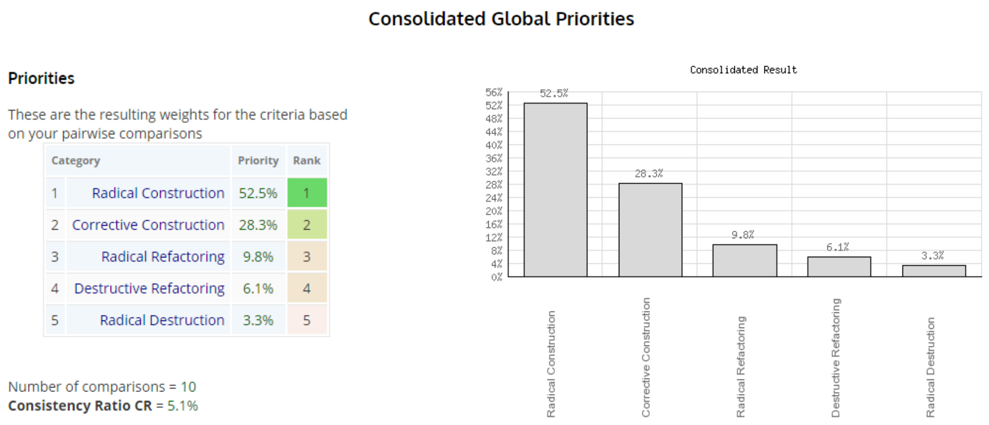
Table 1. AHP Criteria Comparison Matrix

Criteria	Criteria				
	RC	**CC**	**RR**	**DR**	**RD**
Radical Construction (**RC**)	1	3	6	8	9
Corrective Construction (**CC**)	1/3	1	4	6	8
Radical Refactoring (**RR**)	1/6	1/4	1	2	4
Destructive Refactoring (**DR**)	1/8	1/6	1/2	1	3
Radical Destruction (**RD**)	1/9	1/8	1/4	1/3	1

Using the online tool BPMSG AHP Priority Calculator [27], we obtain the priority vector P given in Table 2 (which helps determine the most important criterion). We notice that the BPMSG calculator automatically performs the matrix normalization in the background.

Table 2. Priority Vector with Priority Decimal Values

Criteria	Criteria				
	RC	**CC**	**RR**	**DR**	**RD**
Radical Construction (**RC**)	1.00	3.00	6.00	8.00	9.00
Corrective Construction (**CC**)	0.33	1.00	4.00	6.00	8.00
Radical Refactoring (**RR**)	0.17	0.25	1.00	2.00	4.00
Destructive Refactoring (**DR**)	0.13	0.17	0.50	1.00	3.00
Radical Destruction (**RD**)	0.11	0.13	0.25	0.33	1.00
Priority Vector (P)	**0.5252**	**0.283**	**0.0975**	**0.0615**	**0.0328**


Consolidated Global Priorities

Priorities

These are the resulting weights for the criteria based on your pairwise comparisons

	Category	Priority	Rank
1	Radical Construction	52.5%	1
2	Corrective Construction	28.3%	2
3	Radical Refactoring	9.8%	3
4	Destructive Refactoring	6.1%	4
5	Radical Destruction	3.3%	5

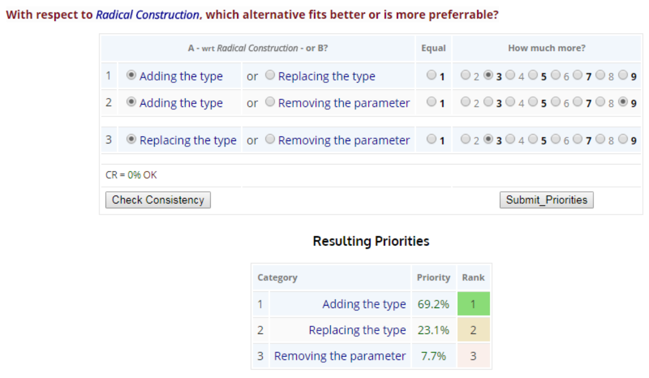
Number of comparisons = 10
Consistency Ratio CR = 5.1%

Fig. 3. Criteria priorities.

Fig. 3, plotted using the BPMSG-AHP priority calculation system, shows the highest score for the “Radical Construction” criterion. Therefore, its purpose of use takes precedence over other criteria.

On the other hand, it is necessary to evaluate all the three alternatives against the five criteria using the AHP technique (Alternatives vs Criteria).

Fig. 4 shows an example comparing three fixes with respect to the "Radical Construction" criterion using the BPMSG AHP priority calculation system. It turns out that the "Radical Construction" criteria in RA.2 (replace the type) and RA.3 (remove the parameter) are less important than RA.1 (add the type).


With respect to *Radical Construction*, which alternative fits better or is more preferrable?

A - wrt *Radical Construction* - or B?				Equal	How much more?
1	[✓ Adding the type]	or	Replacing the type	1	2 [✓ 3] 4 5 6 7 8 9
2	[✓ Adding the type]	or	Removing the parameter	1	2 3 4 5 6 7 8 [✓ 9]
3	[✓ Replacing the type]	or	Removing the parameter	1	2 [✓ 3] 4 5 6 7 8 9

CR = 0% OK

Resulting Priorities

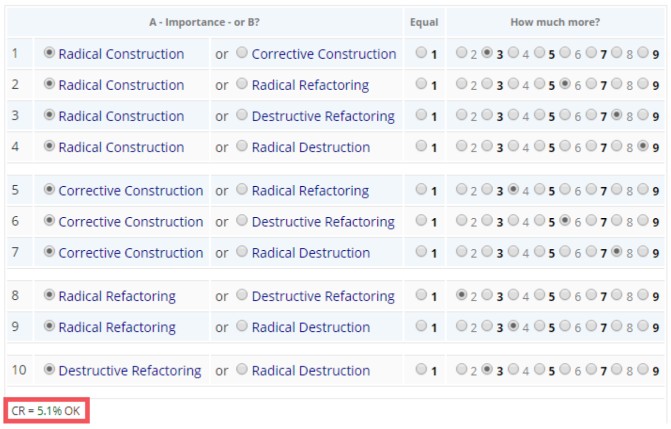
	Category	Priority	Rank
1	Adding the type	69.2%	1
2	Replacing the type	23.1%	2
3	Removing the parameter	7.7%	3

Fig. 4. A comparison of the repair actions in accordance with "Radical Construction" criteria.

Note that we proceed with the same way for the comparison of the three repair actions with the other remaining criteria.

3.3.3. Activity 3

Fig. 5 shows the Consistency Ratio *CR* = 5.1% < 10% for the criteria comparison matrix. To mention that 10% is the upper acceptable limit for *CR*. The values of the matrix are therefore consistent; this means that the initial values considered for different criteria are well defined (i.e. efficiency).



A - Importance - or B?			Equal	How much more?
1	[✓ Radical Construction]	or Corrective Construction	1	2 [✓ 3] 4 5 6 7 8 9
2	[✓ Radical Construction]	or Radical Refactoring	1	2 3 4 5 [✓ 6] 7 8 9
3	[✓ Radical Construction]	or Destructive Refactoring	1	2 3 4 5 6 7 [✓ 8] 9
4	[✓ Radical Construction]	or Radical Destruction	1	2 3 4 5 6 7 8 [✓ 9]
5	[✓ Corrective Construction]	or Radical Refactoring	1	2 3 [✓ 4] 5 6 7 8 9
6	[✓ Corrective Construction]	or Destructive Refactoring	1	2 3 4 5 [✓ 6] 7 8 9
7	[✓ Corrective Construction]	or Radical Destruction	1	2 3 4 5 6 7 [✓ 8] 9
8	[✓ Radical Refactoring]	or Destructive Refactoring	1	[✓ 2] 3 4 5 6 7 8 9
9	[✓ Radical Refactoring]	or Radical Destruction	1	2 3 [✓ 4] 5 6 7 8 9
10	[✓ Destructive Refactoring]	or Radical Destruction	1	2 [✓ 3] 4 5 6 7 8 9

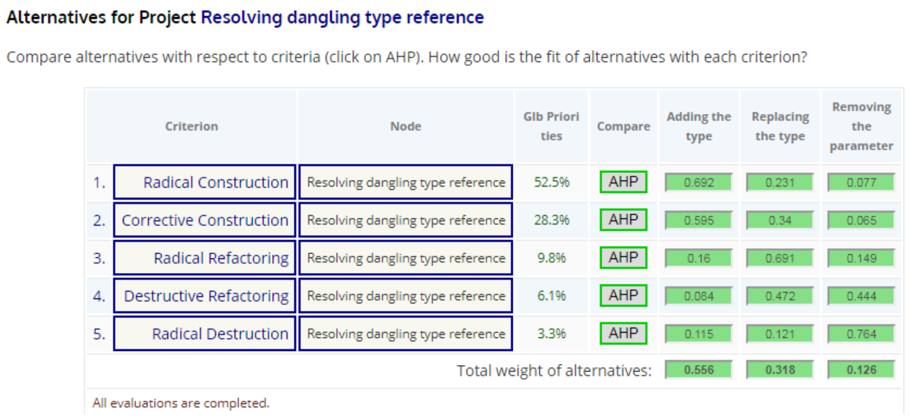
CR = 5.1% OK

Fig. 5. Computation of the consistency ratio.

Similarly, the consistency checking done in the background by the tool shows that all the other comparison matrices (alternatives vs criteria) are efficient (i.e. the CR value is less than 10%).

3.3.4. Activity 4

In this state, we define which of the three repair operations (RA.1) to (RA.3) will be performed. After a synthesis of the final priorities, we obtain the results presented in Fig. 6.



Alternatives for Project Resolving dangling type reference

Compare alternatives with respect to criteria (click on AHP). How good is the fit of alternatives with each criterion?

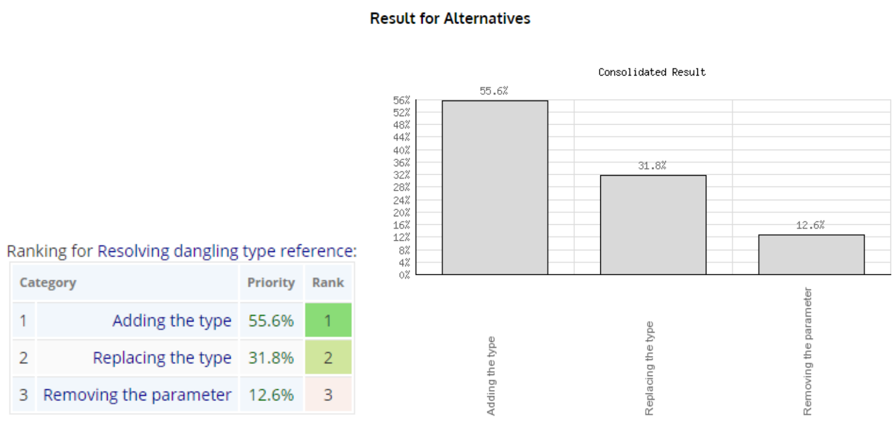
	Criterion	Node	Glb Priorities	Compare	Adding the type	Replacing the type	Removing the parameter
1.	Radical Construction	Resolving dangling type reference	52.5%	AHP	0.692	0.231	0.077
2.	Corrective Construction	Resolving dangling type reference	28.3%	AHP	0.595	0.34	0.065
3.	Radical Refactoring	Resolving dangling type reference	9.8%	AHP	0.16	0.691	0.149
4.	Destructive Refactoring	Resolving dangling type reference	6.1%	AHP	0.084	0.472	0.444
5.	Radical Destruction	Resolving dangling type reference	3.3%	AHP	0.115	0.121	0.764
				Total weight of alternatives:	0.556	0.318	0.126

All evaluations are completed.

Fig. 6. The three alternatives ranking.

More explicitly, as shown in Fig. 7, RA.1 (add the type) has a priority of 55.6%, while RA.2 (replace the type) is 31.8% and RA.3 (remove the parameter) is 12.6%.

As a conclusion, using an AHP-based approach to help modelers choose the correct repair operation to fix "*dangling type reference*" inconsistencies encourages adding types to affected UML models as a repair operation.



Ranking for Resolving dangling type reference:

Category		Priority	Rank
1	Adding the type	55.6%	1
2	Replacing the type	31.8%	2
3	Removing the parameter	12.6%	3

Fig. 7. Graphical representation of the ranking of final repair operations

Applying the same method at different stages of development of the model will show different results:

- "*Promoting Replacing the type* repair action when the modeler has a refactoring vision of the model."
- "*Promoting Removing the parameter* repair action when the modeler has a recast vision of the model."

These results are very promising, considering that the use of the AHP technique in our context is so compelling that it can be used to build frameworks and tools that can be integrated with other inconsistency management solutions.

4. Experimentation and Discussion

To validate the initial assumption concerning the efficiency of the proposed method, we based our experimentation on other examples that concern intra-model inconsistencies between Class and Sequence diagrams. Some of these UML model inconsistencies were initially detected in an academic management system case study presented in one of our previous works [41]. To mention that these UML model inconsistency examples, and many others, are textually expressed as "Consistency Rules" in the exhaustive list presented in [40].

For each of these UML inconsistency examples, we proposed a set of repair actions (the number of alternatives differs for each case) and applied the AHP-based method with the same 5 criteria aforementioned in the illustrative example presented in section 3.1. We also assume for each execution of the method, a different development state and strategy to verify if the category of the chosen repair action will change according to this development state. Table 3 summarizes the repair action recommended for every inconsistency example according to different assumptions concerning the followed modeling strategy.

The obtained results show in each case that this method is in total alignment with the modeling strategy adopted by the modeler since the weights values of the criteria are changing according to that. And then, the proposed alternative (repair action) changes accordingly.

Actually, rather than prescribing the "correct" repair operation, AHP helps the modelers find the repair operation that best suits their goals and their understanding of the inconsistency. The results obtained from the executed cases show the relevance of the method from this point of view. These results are satisfactory as they fully conform to the vision originally presented as a hypothesis; this means that the proposed method is strongly influenced by the modeling strategy and the current modeling stage. For example, for a constructive vision of modeling (i.e. Building strategy), it is often advisable to add elements to the model. This was shown in the case of the majority of the presented examples. For instance, if we have three alternatives each represented by a different type of operations (add, change, and delete). Therefore, the suggested action recommends adding elements to the model; since the followed modeling strategy is constructive-oriente.

Table 3. Summary of the Recommended Repair Actions for Different UML Model Inconsistencies

Inconsistency examples	Repair Actions (RA)	The recommended repair action (using AHP)		
		Considering Assumption 1	Considering Assumption 2	Considering Assumption 3
A. Abstract Object	**A1.** Change the concerned class to "concrete" **A2.** Remove the instantiated object	A1	A1	A2
B. Abstract Operation	**B1.** Make the abstract operation concrete (implement it) **B2.** Change the concrete class to abstract **B3.** Remove the abstract operation from the model	B1	B2	B3
C. Dangling Operation	**C1.** Place the operation in the class attached to the receiving event of the message **C2.** Change the class attached to the receiving event of the message **C3.** Remove the message that refers to the operation	C1	C1	C3
D. Navigation Incompatibility	**D1.** Change the calling direction of the message **D2.** Change the navigation constraint on the association **D3.** Remove the sequence diagram message **D4.** Remove the navigation constraint on the association	D1	D2	D4
E. Connector Type Incompatibility	**E1.** Add an association between the concerned classes **E2.** Change the types of the connected elements **E3.** Remove the connector	E1	E2	E3
F. Multiplicity Incompatibility	**F1.** Change the multiplicity of the association typing the owning connector **F2.** Change the multiplicity of the connector end **F3.** Remove the connector	F2	F2	F3
G. Classless Instance Connectable Element	**G1.** Add a new class to instantiate the object **G2.** Change the type of the object in the sequence diagram	G1	G2	G2
H. Dangling Connectable Feature Reference	**H1.** Add an operation in the class diagram that corresponds to the message **H2.** Change the message to correspond to an existing operation	H1	H2	H2

However, the situation becomes more difficult when we have more alternatives with the same type of action (e.g. three adding actions). In this case, how can we choose the appropriate repair action? The answer to this question is by no means obvious. However, with a better understanding of the inconsistency (i.e. that can be guaranteed by a good diagnosis), the modeler can always set a new target for his processing activity, and then set different/new criteria than the ones used in our experimentation examples. In fact, AHP is a rational and extensible framework for representing and quantifying elements of a decision problem, linking those elements to an overarching goal, and evaluating alternative solutions.

Assumption 1: Constructive/Building strategy **|** **Assumption 2:** Refactoring strategy **|** **Assumption 3:** Recasting/Cleaning strategy

4.1.1. Description of the presented UML model inconsistencies

A. Abstract Object: A model contains an instance specification of an abstract class that does not have any concrete subclasses.

B. Abstract Operation: A class diagram contains an abstract operation that is defined in a concrete class.

C. Dangling Operation: A sequence diagram contains a message that refers to an operation that does not belong to the class attached to the receiving event of the message.

D. Navigation Incompatibility: A sequence diagram contains a message of which calling direction does not match the navigation constraint on the corresponding association

E. Connector Type Incompatibility: A model contains a connector for which: the types of the connectable elements that the ends of the connector are attached to do not conform to the types of the association ends of the association that types the connector. (Messages only between related classes)

F. Multiplicity Incompatibility: A model contains a connector end of which the multiplicity is more general than the multiplicity of the association typing the owning connector.

G. Classless Instance Connectable Element: Using an object in the sequence diagram which is non-existing in the class diagram.

H. Dangling Connectable Feature Reference: Every message in the sequence diagram must correspond to a method in the class diagram.

4.1.2. Threats to validity

Another point to consider when assessing the soundness of this work is the ability of the method to propose alternatives that will not damage the other parts of the model. This point is not guaranteed by our proposition since it concerns other sub-activities of handling inconsistencies namely, managing dependencies between inconsistencies and evaluating the risks and the impact of the chosen repair action. These activities are out of the scope of this paper but could be considered in future work.

From another perspective, the proposed AHP-based solution also has its weaknesses. In fact, besides determining the criterion weights, the extraction and identification of important criteria are the two main steps followed in the AHP technique. Therefore, human interference with these two activities greatly affects the results. The subjectivity of determining criteria or determining their weights is one of the drawbacks that can negatively impact the technical relevance of the proposed solution.

Decision-making, on the other hand, involves ranking alternatives based on their criteria or characteristics. However, when new alternatives are added to the decision problem, the ranking of the old alternatives may change. This situation is called "rank inversion" and is a well-known problem in decision-making techniques.

5. Related Work

As stated in [15], handling inconsistencies can be ensured using different activities including systematically diagnosing inconsistencies and their causes, generating a set of repair actions, evaluating costs and risks, selecting and executing the corresponding actions, in addition to managing dependencies

between inconsistencies. Nevertheless, the relatively few proposals existing in literature are generally focused on identifying necessary repair actions or managing dependencies between inconsistencies. In contrast, our proposed AHP-based method organizes and analyzes the decision making concerning the inconsistency fixing to improve the repair process. Actually, this method focuses more on how to handle the repairs from a management perspective. Thing that is noticed as not taken into granted in the literature.
In what follows, we present a survey of existing work concerning the aforementioned handling activities:

5.1.1. Generating repair actions

Egyed et al. [28] integrate in the design tool IBM Rational Rose a technique that automatically generates a specific set of changes to fix inconsistencies and provides information about the impact of each change on all consistency rules. To deal with the large number of repair actions, Reder and Egyed [29] represent repairs in a linearly growing repair tree as alternatives and sequences reflecting the syntactic structure of the inconsistent design rule. The approach is automated and tool-based and is applicable to arbitrary modeling and constraint languages. With the same perspective, Marchezan et al. [38] propose an approach that focuses on the immediate cause of an inconsistency to generate a repair tree containing a subset of repair actions focused on addressing this root cause. To reduce repair opportunities, they provide ownership-based filters to filter repairs that change model elements that are not owned by the developer. Da Silva et al. [30] provide a step towards the automation of deciding what needs to be done to restore the models consistency. They propose a method that generates possible repair actions for an inconsistent model. The method also tries to avoid generating unused repair actions by configuring the depth of the explored search space. Puissant et al. [31] try to automate the resolution of design model inconsistencies by using the artificial intelligence technique of automated planning. They generate one or more resolution plans for the purpose of resolving such inconsistencies. They implement a regression planner in Prolog called Badger that generates such plans. They also offer the possibility to adjust the presentation order of resolution plans by modifying the cost function of the planner algorithm. Bo Wang et al. [32] provide a dynamic-priority based approach to interactively fix inconsistencies in feature models. The approach uses the constraint hierarchy theory to automatically recommend a solution to fix inconsistencies and supports domain analysts to gradually reach the desirable solution by adjusting priorities of constraints.

5.1.2. Managing dependencies between inconsistencies

Inconsistent resolutions can lead to new inconsistencies. To address this issue, Mens et al. [33] represented inconsistency detection and resolution as graph transformation rules and applied key pair analysis theory to analyze potential dependencies between model inconsistency detection and resolution. Mens and Van Der Straeten [34] exploit the mechanism of critical pair analysis to analyze dependencies and conflicts between inconsistencies and resolutions in addition to the analysis of the completeness of resolutions. With the same objective, Ohrndorf et al. [35], [36] propose possible repairs by tracing unfinished modifications to identify the cause of inconsistencies. Oriol et al. [37] use a logic formalization to automatically compute the repairs of an update. The approach aims to find the minimum additional changes that bring the UML/OCL conceptual schemas to a new state where all constraints are satisfied and to fix up non-executable operations.

6. Conclusion

Inconsistency handling involves many activities, such as identifying possible resolution actions to deal with inconsistencies, evaluating the costs, risks, and benefits of applying those actions, and selecting and implementing a proposed action. In this paper, we focus on helping modelers choose the right solution action for their modeling goals and strategies. To do this, we use the AHP technique to organize and analyze modeling decisions. By using this technique, we implicitly let the modeler specify its various parameters, namely the important criteria and the weights of the criteria. The obtained results are seen as clues that help in making the right decision. Ultimately, it is up to the software designer to take action to fix detected inconsistencies.

We proposed this semi-automated approach, in which the human intervention is pretty crucial, due to the difficulty of totally automating this process. Making a good choice is a matter of well knowing the context and all the circumstances of the model. This is why it is up to the modeler, after all, to decide which action must be performed.

As future work, we aim to work on a novel approach to generate repair actions taking into consideration the elimination of false and non-minimal repairs, in order to provide small, complete and correct repair actions. The provided repair actions set will be the input of our AHP-based proposed method (the alternatives). In addition to that, we intend to dig deeper to find a way on how to mitigate the subjectivity that can result from the human intervention in the proposed AHP-based solution. On the other hand, and since there is no guarantee to have a correct model when applying a repair action. Because the new design may contain new arising inconsistencies resulting from the actions of resolving other inconsistencies. Then, regardless the approach used to resolve inconsistencies, the repairs should not be seen in isolation and the management of these dependencies should be taken into consideration.

Conflict of Interest

The author declares no conflict of interest.

References

[1] Cernosek, G., & Naiburg, E. (2004). A technical discussion of software modeling the value of modeling.

[2] UML 2.5. Retrieved August 22, 2022, from http://www.omg.org/spec/UML/2.5/PDF/

[3] Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *Software Development Process*.

[4] Huzar, Z., Kuzniarz, L., Reggio, G., & Sourrouille, J. L. (2005). *Consistency Problems in UML-Based Software Development*. Springer, Berlin, Heidelberg, 1–12.

[5] Allaki, D., Dahchour, M., & En-Nouaary, A. (2015). A new taxonomy of inconsistencies in UML models with their detection methods for better MDE. *Intl. J. Comp. Sci. Appl.*

[6] MDA. Retrieved from http://www.omg.org/mda/

[7] Lange, C., Chaudron, M. R. V., Muskens, J., Somers, L. J., & Dortmans, H. M. (2003). An empirical investigation in quantifying inconsistency and incompleteness of UML designs. *Work. Consistency Probl. UMLbased Softw. Dev. II*, 26–34.

[8] Pooley, R. J. J., & Stevens, P. (1999). Using UML: Software engineering with objects and rules.

[9] Finkelstein, A., Spanoudakis, G., & Till, D. (1996). Managing interference. *Proceedings of the Second International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development* (pp. 172–174).

[10] Straeten, R. V. D. (2005). Inconsistency management in model-driven engineering. A*n Approach Using Descr. Logics*, 2005.

[11] Allaki, D., Dahchour, M., & En-Nouaary, A. (2016). Detecting and fixing UML model inconsistencies using constraints. *Proceedings of the 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*, 295–299.

[12] Singh, M., Sharma, A. K., & Saxena, R. (2016). Formal transformation of UML diagram: Use case, class, sequence diagram with Z notation for representing the static and dynamic perspectives of system. *Adv. Intell. Syst. Comput.*, *409*, 25–38.

[13] Straeten, R. V. D., Jonckers, V., & Mens, T. (2007). A formal approach to model refactoring and model

refinement. *Softw. Syst. Model.*, *6(2)*, 139–162.

[14] Kalibatiene, D., Vasilecas, O., & Dubauskaite, R. (2013). Ensuring consistency in different IS models – UML case study. *Balt. J. Mod. Comput.*, *1(12)*, 63–76.

[15] Spanoudakis, G., & Zisman, A. (2001). Inconsistency management in software engineering: Survey and open research issues. *1*, 329–380.

[16] Nuseibeh, B., Easterbrook, S., & Russo, A. (2000). Leveraging inconsistency in software development. *Computer (Long. Beach. Calif)*, *33(4)*, 24–29.

[17] Saaty, T. (1980). *The Analytic Hierarchy Process: Planning, Priority Setting, Resources Allocation*. M Cgraw-Hill.

[18] Saaty, T. L. (2001). Decision making with dependence and feedback: The analytic network process: The organization and prioritization of complexity. *370*.

[19] Roy, B. (1968). Classement et choix en présence de points de vue multiples. *La Rev d'Informatique Rech. Opérationelle*, 57–75.

[20] Fontela, A., & Gabus, E. (1976). *The Dematel Observer, Dematel 1976 Report*.

[21] Brans, J. P., Mareschal, B., & Vincke, P. (1984). PROMETHEE: A new family of outranking methods in multicriteria analysis. *Oper. Res.*, 477–490.

[22] Hwang, C., & Yoon, K. (1981). *Multiple Attribute Decision Making: Methods and Applications: A State-of-the-Art Survey*.

[23] Mardani, A., Jusoh, A., Nor, K. M., Khalifah, Z., Zakwan, N., & Valipour, A. (2015). Multiple criteria decision-making techniques and their applications — A review of the literature from 2000 to 2014. *Econ. Res. Istraživanja*, *28(1)*, 516–571.

[24] Görener. (2012). Comparing AHP and ANP: An application of strategic decisions making in a manufacturing company. *Int. J. Bus. Soc. Sci.*, *3(11)*, 194–208.

[25] Matteucci, I., Mori, P., & Petrocchi, M. (2013). Prioritized execution of privacy policies. *Lect. Notes Comput. Sci.*, 133–145.

[26] Saaty, T. L. (1977). A scaling method for priorities in hierarchical structures. *J. Math. Psychol.*, *15(3)*, 234–281.

[27] B. Priority Calculator AHP. Retrieved from http://bpmsg.com/academic/ahp.php

[28] Egyed, A., Letier, E., & Finkelstein, A. (2008). Generating and evaluating choices for fixing inconsistencies in UML design models. *Proceedings of the ASE 2008 - 23rd IEEE/ACM Int. Conf. Autom. Softw. Eng*. (pp. 99–108).

[29] Reder, A., & Egyed, A. (2012). Computing repair trees for resolving inconsistencies in design model. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*.

[30] Silva, M. A. A. D., Mougenot, A., Blanc, X., & Bendraou, R. (2010). Towards automated inconsistency handling in design models. *Lect. Notes Comput. Sci.,* 348–362.

[31] Pinna, J., Puissant, R., Van, D. S., & Mens, T. (2015). Resolving model inconsistencies using automated regression planning. *Softw. Syst. Model.*, *14(1)*, 461–481.

[32] Wang, B., Xiong, Y. F., Hu, Z. J., Zhao, H. Y., Zhang, W., & Mei, H. (2014). Interactive inconsistency fixing in feature modeling. *J. Comput. Sci. Technol.*, *29(4)*, 724–736.

[33] Mens, T., Straeten, R. V. D., & Hondt, M. (2006). Detecting and resolving model inconsistencies using transformation dependency analysis. *Model Driven Eng. Lang. Syst.*, 200–214.

[34] Mens, T., & Straeten, R. V. D. (2007). Incremental resolution of model inconsistencies. *Recent Trends Algebr.*, 111–126.

[35] Ohrndorf, M., Pietsch, C., Kelter, U., & Kehrer, T. (2018). ReVision: A tool for history-based model repair recommendations. *Proceedings of the 40th International Conference on Software Engineering:*

Companion Proceeedings, Association for Computing Machinery.
[36] Ohrndorf, M., Pietsch, C., Kelter, U., Grunske, L., & Kehrer, T. (2021). History-based model repair recommendations. *ACM Trans. Softw. Eng. Methodol*.
[37] Oriol, X., Teniente, E., & Tort, A. (2015). Computing repairs for constraint violations in UML/OCL conceptual schemas. *Data Knowl. Eng*., *99*, 39–58.
[38] Marchezan, L., Kretschmer, R., Assunção, W. K. G. *et al*. (2022). Generating repairs for inconsistent models. *Software and Systems Modeling*, 1-33.
[39] Allaki, D., Dahchour, M., & En-Nouaary, A. (2018). An AHP-based method to fix inconsistencies in UML collaborative modeling. *Proceedings of the 2018 IEEE 5th International Congress on Information Science and Technology*.
[40] Torre, D., Labiche, Y., & Genero, M. (2014). UML consistency rules: A systematic mapping study. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*.
[41] Allaki, D., Dahchour, M., & En-Nouaary, A. (2017). Building consistent UML models for better model driven engineering. *Journal of Digital Information Management*, *15(5)*.
[42] Allaki, D., Dahchour, M., & En-Nouaary, A. (2017). Managing Inconsistencies in UML models: A systematic literature review. *Journal of Software*, *12(6)*, 454-471.