CACiC 2023
CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN

Conflict resolution for aspect-oriented static modeling

Fernando Pinciroli[1], Laura Zeligueta[2] and Marcelo Palma[2]

[1] Instituto de Investigaciones, Universidad Nacional de San Juan, Argentina
fpinciroli@iinfo.unsj.edu.ar

[2] Research Institute, Universidad Champagnat, Mendoza, Argentina
{zeliguetalaura, mpalma}@uch.edu.ar

Abstract. Separation of concerns allows the achievement of important benefits in all phases of the software development life cycle. Thus, it is possible to take advantage of this technique with the consequent improvement of the understanding of the models. However, conflicts of different types may be produced when concerns are composed to form a single system, due to the same fact of having them managed separately. Additionally, this problem is increased by the number of people needed to deal with large projects. This article is focused on the composition of concerns in structural models, in which each concern is realized by an individual class diagram. In this paper, we present an experience in which four systems analysts elaborated six class diagrams which belong to a single system, and we expose the conflicts that occurred after the composition of the diagrams. After the analysis and classification of the conflicts, a set of modeling agreements and recommendations was elaborated in order to reduce them. Then, the models were rebuilt and composed again, with a significant decrease in the number of conflicts detected after the second composition.

Keywords: aspect-oriented models; class models; model composition; conflict resolution.

1. Introduction

The aspect-oriented paradigm offers a series of benefits that are described in a vast literature [1] [2] [3], and there is even a great amount of studies which show that, in the cases where this paradigm was used in real-world settings, those benefits were indeed achieved [4]. In addition, this approach solves the problems of scattering and tangling in the source code developed with object-oriented programming [5].

All the advantages are achieved due to modularization, a very important principle of software engineering [6] [7] where the aspect-oriented approach makes the most of keeping the crosscutting concerns separate throughout the entire software development life cycle [7]. Separation of concerns allows the achievement of these benefits not only in the programming phase, but also in all phases of the life cycle [8] [9]. Thus, it is possible to take advantage of the benefits of the reduction of size in software models, with the consequent improvement of the understanding of the models [10].

It might be the case that separation of concerns may generate conflicts of different types when concerns are composed to form a single system, as a result of having managed the concerns separately [11] [12] [13]. And this risk is increased by the number of people needed to deal with large projects.

In this article, we study the conflicts that arise after the composition of concerns in structural models. These models are generally made with class diagrams, and each

concern is realized by an individual class diagram [14]. Every concern usually contains classes that are also present in other diagrams, although with different members and relationships, due to the specific nature of the concern they realize. In a system of medium to high complexity, the intervention of several analysts is required. However, due to the different working styles they can have, many other conflicts may take place when all diagrams are composed into only one. We present an experience in which four systems analysts elaborated six class diagrams of a same system and we present the conflicts which occurred after the composition of the diagrams. Research in the field of software engineering requires the empirical study of phenomena that take place in the real world and, for this work, we have used a qualitative method called post-mortem analysis [15].

In Section 2, we briefly describe the process we have followed to develop the class models individually. Section 3 presents the strategy we chose to compose the models for the first time. In Section 4 we discuss the findings, focusing on the main conflicts detected after the first composition. Section 5 offers the modeling agreements which emerged after the analysis and classification of conflict types detected and the analysts used to rebuild their models. In Section 6 we present the new results after the second composition and, in Section 7, we provide the final results by comparing the conflicts between both compositions, with and without the modeling agreements. Finally, Section 8 presents our conclusions and the final thoughts with regard to future work.

2. Elaboration of class models

In order to conduct this study in the software development life cycle stage corresponding to static modeling, four systems analysts were summoned, each of which developed four class diagrams corresponding to three processes that belong to a biochemical laboratory. These three processes were selected from among the most complex of the 85 processes automated by the software system under study: manage protocol, transfer sample and receive sample.

Three analysts drew up one diagram each, while the fourth analyst, who we shall refer to as “expert analyst”, given his vast experience in the problem domain, drew up diagrams of the three processes. This distribution allowed us to compare the difference in the number of conflicts that occur if the diagrams are made by a single person or by a team. It was also ensured that each diagram has been developed by two analysts, to observe the conflicts that arise when models are built from two different points of view.

Each class diagram performs a single functional software requirement, in order to keep concerns separate throughout the entire life cycle, as proposed by various authors [14] [16] [17]. In the symmetric aspect-oriented approach, each model is equally important, as, as opposed to the asymmetric approach, in which a base portion of the system is considered to be affected by crosscutting concerns [18] [19].

After individual modeling, we obtained six class diagrams, and the number of classes shown in Table 1. The differences in the number of classes already suggest the appearance of a significant number of conflicts.

Table 1. Classes obtained.

Analyst	Concern		
	Manage protocols	Transfer samples	Receive samples
#1	50		
#2		23	
#3			18
#4	38	32	29

The analyst #4 -the expert analyst- modeled 99 classes to build the three class diagrams used for the experience. The entire model of the laboratory is made up of 202 classes, of which the concerns selected in this work cover approximately 50% of the whole model.

3. First composition of the models

The purpose of this work is to study the conflicts that may occur when several analysts intervene in the production of different parts of the static modeling within the same system, at the abstraction level of software requirements. Thus, we have produced a single model by applying a composition process on the six class diagrams developed by the different analysts.

The composition techniques can be classified into two categories: specification-based and heuristic-based techniques [20]. In specification-based techniques, the analysts explicitly specify the relationships among the elements of the source models before being composed [21] [22]; in heuristic-based techniques, the relationships are guessed by a set of predefined heuristics.

There are four proposed heuristic-based composition techniques: merge [14] [17] [23], override [14] [17] [24], select [14] and union [20]. In our work, we applied merge, but we made a variation to the original merge technique. Instead of overlapping the members when they have the same names, we duplicated them if they have different data types, which in turn avoids the “structural superimposition problem” highlighted by Tian et al. [25]. As a result, with this new alternative of composition, we now have five forms of composition: merge by overlapping, merge by duplication (the new one), override, select and union.

Manual composition is complicated and susceptible to human error [24], but we wanted to conduct it that way in order to accurately analyze every conflict or matching, classify each conflict and be able to design standard solutions to prevent or, at least, reduce their appearance.

Although the composition was carried out manually and guided by the expert analyst, we used Enterprise Architect to facilitate the detection of inconsistencies and avoid errors.

The composition process was as follows:

- We first composed 3 class diagrams out of the ones created by the expert analyst who drew up all the models. Some conflicts occurred at this stage, but all of them

were due to human error which, once corrected, resulted in a single, perfectly composed model, without conflicts.

- Then, we added the three remaining diagrams, one by one, by using the technique called "element matching" [26], and looked for elements with matching signatures to be composed.
- Every time we composed an element or a member, we updated the statistics.

The criteria for class composition were:

- We took a class and looked for it in the model by signature: name and {isAbstract} [27]. If it already existed, we considered it a "matching class", if not, an "added class".
- In the case of an "added class", we analyzed if it already existed in the model with a different name or if it did not exist at all. In the first case, we considered it, then, as a "conflicting class"; otherwise, we considered the class arose simply due to a different analysis.

The criteria for attribute composition were:

- All the attributes of the "added classes" were considered as arising from a different analysis.
- The attributes of the matching classes were classified into three types:
  - "Matching attribute": those whose signatures completely match [27]: name, data type, visibility and properties.
  - "Conflicting attribute": those that match in name but not in type, or that do not match in name but represent the same concept as another existing attribute.
  - "Added attribute": those that simply come from a different analysis.

Once the composition was completed, we obtained the results we shall present in the next section. There, we also analyze the causes of the production of added and conflicting classes and attributes.

4. Conflicts detected after the first composition

According to what was mentioned in Section 3, the modeling of the expert analyst who developed all the models should not have conflicts, except for some human errors we had already detected when reviewing the resulting models and that could easily be corrected. According to Bussard et al. [28] this strategy could prevent inherent conflicts, which can be also classified as semantic conflicts [25].

However, the composition of the models developed by the rest of the analysts did present different types of conflicts, which we describe below. All the conflicts detected could be classified as semantic conflicts.

Conflict #1: the same concept modeled in different ways. This is one kind of "redundancy" [29]. The analysts modeled the same reality in different ways, so different classes appeared in the composed model to represent the same concept. Table 2 shows the number of different classes, "added classes", with respect to the one designed by the expert analyst, and how many of them correspond to a different modeling style. The "Percentage of total" column indicates the fraction of added classes with respect to the total of classes. The "Percentage on added classes" column indicates the fraction of

classes that appeared due to a different modeling with respect to the total of added classes.

Table 2. Conflicts due to "added classes".

Analyst	Total classes	Added classes	Percentage of total	Different modeling	Percentage on added classes
#1	50	41	82.00%	25	60.98%
#2	18	13	72.22%	12	92.31%
#3	23	14	60.87%	3	21.43%
Total	**91**	**68**	**74.73%**	**40**	**58.82%**

Conflict #2: The same class named differently. This conflict can also be classified as "redundancy" (Table 3). Since the composition is made by combining elements with the same names, another conflict occurs when analysts use different names to label the same concepts. We call them "conflicting classes". France et al. propose the use of "pre-merge directives" to solve this issue [24].

Table 3. Conflicts due to "conflicting classes".

Analyst	Different classes	Different naming	Percentage
#1	41	16	39.02%
#2	13	1	7.69%
#3	14	11	78.57%
Total	**68**	**28**	**41.18%**

Conflict #3: Different classes with the same name. The other side of the conflict mentioned in the previous point corresponds to the use of the same name to name different classes. We did not detect any such cases, categorized as "naming conflicts on aspects" [25] or as "inconsistency" [29].

Conflict #4: Attributes of different classes. The different classes from the analysts have attributes that could be representing the same concepts, but which cannot be composed because they belong to different classes. Table 4 shows the number of attributes that cannot be composed.

Table 4. Conflicts due to "added attributes" belonging to "added classes".

Analyst	Total attributes	Attributes of different classes	Percentage
#1	99	72	72.73%
#2	65	51	78.46%
#3	61	33	54.10%
Total	**225**	**156**	**69.33%**

Conflict #5: The same attribute modeled in different ways. Again, this is another kind of "redundancy" type. The analysts modeled the same reality in different ways, so different attributes appear in the composite model to represent the same concept. Table 5 shows the number of attributes other than those specified by the expert analyst, and how many of them correspond to a different modeling style.

Table 5. Conflicts due to "added attributes" belonging to "matching classes".

Analyst	Different attributes	Different modeling	Percentage
#1	22	10	45.45%
#2	14	6	42.86%
#3	28	5	17.86%
Total	**64**	**21**	**32.81%**

Conflict #6: The same attribute named differently. This is another conflict of "redundancy". Since the composition is made by combining elements with the same name, another conflict occurs when analysts use different names to label the same concepts (attributes). This type of conflict is called "naming conflicts on object-oriented components" by Tian et al. [25]. Table 6 shows the number of identical attributes belonging to identical classes, but to which different names were assigned.

Table 6. Conflicts due to "conflicting attributes".

Analyst	Different classes	Different naming	Percentage
#1	22	12	60.98%
#2	14	8	92.31%
#3	28	23	21.43%
Total	**64**	**43**	**67.19%**

Conflict #7: Different attributes with the same name. As with classes, it would be possible for analysts to name different attributes with the same name. We also did not find this kind of inconsistency present in our composed models.

Conflict #8: Different styles of relationships. Because of the number of conflicts between classes are very high at this point, we considered it did not make sense to analyze conflicts in relationships among classes, an analysis we hope to carry out once we are able to reduce conflicts between classes.

5. Modeling agreements and second composition

Most conflict resolution methods require formal specifications [29] [30] [31] [32], while Sardinha et al. offer a tool that does not require formality but only detects conflicts in specifications of high-level requirements and not in classes [31]. Based on the types of conflicts detected, we developed a set of modeling agreements to avoid them or, at least, to reduce them. This experience was originally performed in Spanish and, as a result, many of the solutions are specific to that language. Undoubtedly, it will be necessary to adapt the modeling agreements to every different language.

In general, conflicts can be classified into syntactic and semantic categories [20]. In Table 7 we present the recommendations we elaborated to solve the conflicts that were detected after the first composition.

Table 7. Modeling agreements.

Scope	Type	Rule	Explanation
General	Syntactic	UML standard	UML 2.5.1 standard must be strictly followed.
Class	Semantic	Class instead of attribute	All attributes that could be modeled as a class must be represented that way.
	Syntactic	Association class	Model the association-class as a single class and add the association-class constraint to it.
	Semantic	Many-to-many class	The team must agree on a suitable name; if there isn't one, name the many-to-many class with the two names of the connected classes, in alphabetical order.
	Semantic	Generalization or "Type" class	Use generalization when the subtypes require different members or relationships among them. Use "Type" class when subtypes are not different among them or if they are just a list.
	Syntactic	Use of prepositions	Avoid the use of prepositions in class names.
	Syntactic	Use of accent mark	Avoid the use of accent marks (for languages other than English).
	Semantic	Use of adjectives	Avoid adjectives if possible. Use them if strictly necessary.
	Semantic	Use of plural	Avoid the use of plural names. Use them only if strictly necessary.
	Syntactic	Standard characters	Use only standard characters in class names.
	Syntactic	CamelCase	Class names must follow the CamelCase pattern.
	Semantic	Record	Add the word "Record" as a suffix to name classes with historical elements as objects (it could be a prefix in languages other than English).
	Semantic	Classes with items	Add the word "Item" as a suffix to name classes with items (it could be a prefix in languages other than English).
Attribute	Semantic	Data types	Define a standard set of data types: int, bool, char, etc.
	Semantic	Date and time	Use separate attributes for each concept.
	Semantic	Logical erase	Use the name "isActive" to indicate logical deletion.
	Semantic	Id#	Avoid the use of Id# attributes.
	Semantic	Derived attributes	Avoid the use of derived attributes.
	Semantic	Use of prepositions	Avoid the use of prepositions in attribute names.
	Syntactic	Use of accent mark	Avoid the use of accent marks (for languages other than English).
	Semantic	Use of adjectives	Avoid adjectives if possible. Use them if strictly necessary.
	Semantic	Use of plural	Avoid the use of plural names. Use them only if strictly necessary.
	Syntactic	Standard characters	Use only standard characters in attribute names.
	Syntactic	camelCase	Attribute names must follow the camelCase pattern.
	Semantic	Boolean attributes	Use a verb as a prefix in boolean attributes.
	Semantic	Completeness	Check all the attributes have data type.
	Semantic	Properties	Can be used as usual.

Scope	Type	Rule	Explanation
	Semantic	Standard names	Use standard names for the most common attributes. i.e.: - "name": for the object name. - "number": for the object number. - "description": for the explanation of the meaning of the object. - "comments": for comments, observations, etc. - Suffixes "From" and "To" for ranges. - "value": for value, price, etc. - "date": for the object creation date. - "surname": for family name. "name": in singular, for names.
Relation-ship	Semantic	Generalization	Can be used as usual, but the generalization: - must follow Liskow's principle. - must be checked against cyclic and conflicting inheritance and conflicting to avoid the "object-oriented composition problem [25]. will be flattened before the composition and abstract classes will disappear.
	Syntactic	Generalization lines	Join the generalization lines when the subtypes belong to the abstraction.
	Semantic	Composition and aggregation	Can be used as recommended in UML.
	Semantic	"1..1" multiplicity	Check if it can be "1..*" with the question: "Do we need history?".
	Semantic	"1..*" multiplicity	Check if it can be "0..*" with the question: "Can we find some case where the multiplicity can be 0?".
	Syntactic	Unknown multiplicity	Leave blank when the multiplicity is unknown.
	Semantic	Navigation	Can be used as usual.
	Semantic	Use of prepositions	Avoid the use of prepositions in relationship names.
	Syntactic	Use of accent mark	Avoid the use of accent marks (for languages other than English).
	Semantic	Use of adjectives	Avoid adjectives if possible. Use them if strictly necessary.
	Semantic	Use of plural	Avoid the use of plural names. Use them only if strictly necessary.
	Syntactic	Standard characters	Use only standard characters in relationship names.
	Syntactic	CamelCase	Class names must follow the CamelCase pattern.
	Semantic	Unary relationship	Use role names at each end of the relationship.
	Semantic	Constraints	Can be used as usual.

After our first composition, these modeling agreements were shared and explained within the team. Then, the analysts corrected their previous six class diagrams by applying the model agreements. These diagrams were composed again, following the same strategy:

- The composition technique was merged by duplication.
- We first composed the 3 class diagrams of the expert analyst.
- Then, we added the three remaining diagrams, one by one.
- Every time we composed an element or a member, we updated the statistics.
- Finally, we compared the statistics of both compositions.

6. Conflicts detected after the second composition

In this section we present the results obtained for every type of conflict after the second composition.

Conflict #1: The same concept modeled in different ways. The analysts modeled the same reality in different ways. As a result, different classes appeared in the composed model to represent the same concept. Table 8 shows the number of different classes, namely "added classes", with respect to those designed by the expert analyst, and how many of them correspond to a different modeling style. The "Percentage on added classes" column indicates the rate of classes due to a different modeling with respect to the total of classes that were added.

Table 8. Conflicts due to "added classes".

Analyst	Total classes	Added classes	Percentage of total	Different modeling	Percentage on added classes
#1	50	31	59.62%	28	90.32%
#2	18	13	72.22%	12	92.31%
#3	23	5	21.74%	3	60.00%
Total	**91**	**49**	**44.55%**	**43**	**87.76%**

Conflict #2: The same class named differently. Since the composition is made by combining elements with the same names, another conflict occurs when analysts use different names to label the same concepts. We shall refer to them "conflicting classes" (Table 9).

Table 9. Conflicts due to "conflicting classes".

Analyst	Different classes	Different naming	Percentage
#1	31	3	9.68%
#2	13	1	7.69%
#3	5	2	40.00%
Total	**49**	**6**	**12.24%**

Conflict #3: Attributes of different classes. The different classes among the analysts have attributes that could be representing the same concepts, but they cannot be composed because they belong to different classes (Table 10).

Table 10. Conflicts due to "added attributes" belonging to "added classes".

Analyst	Total attributes	Attributes of different classes	Percentage

#1	140	72	51.43%
#2	67	55	82.09%
#3	61	10	16.39%
Total	**268**	**137**	**51.12%**

Conflict #4: The same attribute modeled in different ways. The analysts modeled the same reality in different ways, so different attributes appear in the composite model to represent the same thing. Table 11 shows the number of attributes other than those of the expert analyst, and how many of them correspond to a different modeling style.

Table 11. Conflicts due to "added attributes" belonging to "matching classes".

Analyst	Different attributes	Different modeling	Percentage
#1	33	29	87.88%
#2	5	3	60.00%
#3	11	8	72.73%
Total	**49**	**40**	**81.63%**

Conflict #5: The same attribute named differently. Since the composition is made by combining elements with the same names, another conflict occurs when analysts use different names to label the same concepts (attributes). Table 12 shows the number of identical attributes belonging to identical classes, but to which different names were assigned.

Table 12. Conflicts due to "conflicting attributes".

Analyst	Different classes	Different naming	Percentage
#1	22	4	12.12%
#2	14	2	40.00%
#3	28	3	27.27%
Total	**64**	**9**	**18.37%**

Conflict #6: Redundancy conflicts. We were able to confirm the need to perform an activity after composing, which is called "post-merge" by different authors [24] [27].

In our experiment, non-derived attributes in their respective source diagrams became derived attributes when they were combined by the merging process. For example, "age" and "date of birth" together make the former derive from the latter. Then, after the composition of the class diagrams, we see as necessary to carry out a new activity to correct the redundancy conflicts that may have arisen.

7. Final results

After the second composition, we collected the differences found with respect to the first composition. It is possible to observe a substantial improvement after the application of the modeling agreements in the production of the models elaborated independently by the different analysts.

Conflict #1: The same concept modeled in different ways. Table 13 compares Tables 2 and 8 and shows that the number of added classes was reduced by 58.62%. We also obtained a 32.97% improvement because these classes are produced by a difference in the analysts' modeling style, which went from 58.85% to 87.76%. We believe that it would not be possible to improve this number with rules, but analysts should have prior training to align their modeling styles.

Table 13. Improvement after the second composition in "added classes".

Composition	Added classes	Different modeling
1st composition	74.73%	58.85%
2nd composition	44.55%	87.76%
Improvement	**58.62%**	**32.97%**

Conflict #2: The same class named differently. Table 14 compares Tables 3 and 9 and shows a significant improvement in the reduction of conflicts due to the different naming criteria of the classes, which were unified by the rules of the modeling agreement.

Table 14. Improvement after the second composition in "conflicting classes".

Composition	Different naming
1st composition	41.18%
2nd composition	12.24%
Improvement	**70.26%**

Conflict #3: Attributes of different classes. Table 15 compares Tables 4 and 10 and presents the logical reduction due to the lower number of added classes, as commented in point 7.1.

Table 15. Improvement after the second composition in "added attributes" belonging to "added classes".

Composition	Attributes of different classes
1st composition	69.33%
2nd composition	51.12%
Improvement	**26.27%**

Conflict #4: The same attribute modeled in different ways. As previously mentioned, the decrease of conflicts due to differences in conventions means that these differences lie in the diverse modeling styles of the analysts involved in the project, as can be seen in Table 16, which compares Tables 5 and 11.

Table 16. Improvement after the second composition in "added attributes" belonging to "matching classes".

Composition	Different attribute modeling
1st composition	32.81%
2nd composition	81.63%
Improvement	**59.80%**

Conflict #5: The same attribute named differently. Table 17 shows, once again, a significant improvement in the reduction of conflicts due to the different naming criteria of the attributes, which were unified by the rules of the modeling agreement. It compares Tables 6 and 12.

Table 17. Improvement after the second composition in "conflicting attributes".

Composition	Different attribute naming
1st composition	67.19%
2nd composition	18.37%
Improvement	**72.66%**

Finally, the matching classes and attributes increased notably. Table 18 summarizes our findings on this issue. The percentages of the matching classes and attributes were calculated with respect to the total of classes and attributes respectively.

Table 18. Matching classes and attributes.

Composition	Matching classes	Matching attributes
1st composition	23 out of 128	5 out of 225
2nd composition	44 out of 110	82 out of 268
Improvement	**55.08%**	**92.74%**

8. Conclusions

We have mentioned that separation of concerns throughout the software development life cycle has the possibility that conflicts of different types may be generated when concerns are composed to form a single system, given the fact that the concerns have been managed separately.

Since each concern is realized by an individual class diagram, all of them have shared classes, although with different members and relationships, due to the specific nature of the concern they realize. Therefore, in Section 4 we have described the conflicts that arose when all the diagrams were composed into only one due to the different working styles performed by different analysts. Thus, throughout our experiment, we were able to demonstrate that the need to establish modeling criteria and rules is critical, in order to reduce the amount and types of conflicts that occur when different analysts intervene to produce, separately, the models that will end up being composed in only one.

This article is part of a larger research project [37] where a series of research is being caried out concerning a framework process that has been developing for several years [36] [37] [38], although there are still many lines open for future work.

Regarding this, we believe there is a lot to do about pre and post composition activities [28]. We are thinking about deepening the study on the impact of changes we do during the composition: addition, removal, modification and derivation of elements in the composed model [21]. The production of modeling patterns will also be very useful. Rules and conventions for naming elements are crucial since the names of the modeling elements are the basis of composition. The study of the different conflict types and their classification will also be a great contribution to design techniques that allow them to be addressed and solved more effectively. The composition of relationships is also an important area to explore [30]. Finally, the automatization of modeling agreements, and the use of online collaborative model editors, will be of great help to obtain an efficient model composition.

References

[1] FanJiang Y, Kuo J, Ma S, Huang W (2010) An Aspect-Oriented Approach for Mobile Embedded Software Modeling. In: Taniar D, Gervasi O, Murgante B, Pardede E, Apduhan BO (eds.) Computational Science and Its Applications, ICCSA 2010, vol. 6017, Berlin, Heidelberg, pp. 257–272. doi: 10.1007/978-3-642-12165-4_21.

[2] Jalali A (2015) Static Weaving in Aspect Oriented Business Process Management. Conceptual Modeling, Stockholm, pp. 548–557.

[3] Singh N, Singh Gill N (2012) Towards an Integrated AORE Process Model for Handling Crosscutting Concerns. IJCA, vol. 37, no. 3, pp. 18–24. doi: 10.5120/4587-6525.

[4] Pinciroli F, Barros Justo JL, Forradellas R (2020) Systematic mapping study: On the coverage of aspect-oriented methodologies for the early phases of the software development life cycle. Journal of King Saud University - Computer and Information Sciences. doi: 10.1016/j.jksuci.2020.10.029.

[5] Kiczales G et al. (1997) Aspect Oriented Programming. Proceedings of the European Conference on Object-Oriented Programming (ECOOP), vol. 1241, p. 25.

[6] Parnas DL (1972) On the criteria to be used in decomposing systems into modules. Commun. ACM, vol. 15, no. 12, pp. 1053–1058. doi: 10.1145/361598.361623.

[7] Dijkstra EW (1982) Selected writings on computing: a personal perspective. Springer-Verlag, New York.

[8] Ye S, He C (2013) A comparison of methods for identification of early aspects. Proceedings of 2013 3rd International Conference on Computer Science and Network Technology, Dalian, pp. 275–279. doi: 10.1109/ICCSNT.2013.6967112.

[9] Rashid A, Moreira A (2006) Domain Models Are NOT Aspect Free. In: Nierstrasz O, Whittle J, Harel D, Reggio G (eds.) Model Driven Engineering Languages and Systems, vol. 4199, Berlin, Heidelberg, pp. 155–169. doi: 10.1007/11880240_12.

[10] Rashid A et al. (2010) Aspect-Oriented Software Development in Practice: Tales from AOSD-Europe. Computer, vol. 43, no. 2, pp. 19–26. doi: 10.1109/MC.2010.30.

[11] Vanoli VL, Marcos CA (2007) Early Conflicts: Análisis y Resolución de Conflictos Tempranos. XIX Encuentro Chileno de Computación, Jornadas Chilenas de Computación, Iquique, p. 1-14.

[12] Sardinha A, Araújo J, Moreira A, Rashid A (2010) Conflict Management in Aspect-Oriented Requirements Engineering. Information Sciences and Technologies Bulletin of the ACM Slovakia, vol. 2, no. 1, pp. 56-59.

[13] Pryor JL, Marcos C (2003) Solving Conflicts in Aspect-Oriented Applications. Proceedings of the Fourth Argentine Symposium on Software Engineering (ASSE´2003), Jornadas Argentinas de Informática e Investigación Operativa, vol. 32, Buenos Aires, pp. 1-10.

[14] Pinciroli F (2020) Proceso marco orientado a aspectos en las etapas tempranas del ciclo de vida del desarrollo de software para una transición en la industria. PhD tesis, Universidad Nacional de San Juan, San Juan.
[15] Wohlin C, Höst M, Henningsson K (2006) Empirical Research Methods in Web and Software Engineering. In: Mendes E, Mosley N (eds.) Web Engineering, Springer-Verlag, Berlin/Heidelberg, pp. 409–430. doi: 10.1007/3-540-28218-1_13.
[16] Jacobson I, Ng PW (2005) Aspect-oriented software development with use cases. Upper Saddle River, Addison-Wesley.
[17] Clarke S, Baniassad E (2005) Aspect-oriented analysis and design: the theme approach. Upper Saddle River, Addison-Wesley.
[18] Bálik J, Vranić V (2012) Symmetric aspect-orientation: some practical consequences. Proceedings of the 2012 workshop on Next Generation Modularity Approaches for Requirements and Architecture, NEMARA '12, Potsdam, p. 7. doi: 10.1145/2162004.2162007.
[19] Reddy YR et al. (2006) Directives for Composing Aspect-Oriented Design Class Models. In: Rashid A, Aksit M (eds.) Transactions on Aspect-Oriented Software Development I, vol. 3880, Berlin, Heidelberg, , pp. 75–105 . doi: 10.1007/11687061_3.
[20] Farias de Oliveira KS (2012) Empirical Evaluation of Effort on Composing Design Models. Pontificia Universidade Católica do Rio de Janeiro, Rio de Janeiro.
[21] Kolovos D, Rose L, García-Domínguez A, Paige RF (2022) The Epsilon Book. https://www.eclipse.org/epsilon/doc/book/EpsilonBook.pdf. Accessed 7 March 2022.
[22] Whittle J, Jayaraman P, Elkhodary A, Moreira A, Araújo J (2009) MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation. In: Katz S, Ossher H, France R, Jézéquel JM (eds.) Transactions on Aspect-Oriented Software Development VI, vol. 5560, Berlin, Heidelberg, pp. 191–237. doi: 10.1007/978-3-642-03764-1_6.
[23] Object Management Group: OMG® Unified Modeling Language® version 2.5.1. OMG® Unified Modeling Language®. https://www.omg.org/spec/UML/2.5.1/PDF. Accessed 7 March 2022.
[24] France R, Fleurey F, Reddy R, Baudry B, Ghosh S (2007) Providing Support for Model Composition in Metamodels. 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), Annapolis, pp. 253–253. doi: 10.1109/EDOC.2007.55.
[25] Tian K, Cooper K, Zhang K, Yu H (2009) A Classification of Aspect Composition Problems. 2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement, Shanghai, pp. 101–109. doi: 10.1109/SSIRI.2009.33.
[26] Kienzle J, Al Abed W, Jacques K (2009) Aspect-oriented multi-view modeling. Proceedings of the 8th ACM international conference on Aspect-oriented software development, AOSD '09, Charlottesville, p. 87. doi: 10.1145/1509239.1509252.
[27] Reddy R, France R, Fleuery F, Baudry B (2005) Model Composition - A Signature based approach. Proceedings Aspect Oriented Modeling Workshop, MODELS/UML, Montego Bay.
[28] Bussard L, Carver L, Ernst E, Jung M, Robillard M, Speck A (2000) Safe Aspect Composition. European Conference on Object-Oriented programming, ECOOP 2000, Sophia, Antípolis and Cannes.
[29] Elasri H, Elabbassi E, Abderrahim S (2018) Semantic integration of UML class diagram with semantic validation on segments of mappings, arXiv:1801.04482 [cs.SE].
[30] Laney R, Barroca L, Jackson M, Nuseibeh B (2004) Composing requirements using problem frames. Proceedings of the 12th IEEE International Requirements Engineering Conference, Kyoto, pp. 113–122. doi: 10.1109/ICRE.2004.1335670.
[31] Mostefaoui F, Vachon J (2007) Design-Level Detection of Interactions in Aspect-UML Models Using Alloy. JOT, vol. 6, no. 7, p. 137. doi: 10.5381/jot.2007.6.7.a6.

[32] Weston N, Chitchyan R, Rashid A (2008) A Formal Approach to Semantic Composition of Aspect-Oriented Requirements. 6th IEEE Intl. Req. Eng. Conference, Barcelona, pp. 173–182. doi: 10.1109/RE.2008.42.

[33] Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology, vol. 64, pp. 1–18. doi: 10.1016/j.infsof.2015.03.007.

[34] Pinciroli F, Zeligueta L, Palma M, Cappello I, Motta E (2021) Desarrollo de incumbencias en el modelado de la vista estática. XXIII Workshop de Investigadores en Ciencias de la Computación, WICC 2021, Chilecito.

[35] Pinciroli F (2015) AOP4ST – Aspect-Oriented Process for a Smooth Transition. Proceedings of the XVII Workshop de Investigadores en Ciencias de la Computación, WICC 2015, Salta, p. 5.

[36] Pinciroli F, Barros Justo JL (2017) Early aspects in 'Aspect-Oriented Process for a Smooth Transition'. XIV Workshop Ingeniería de Software (WIS), Congreso Argentino de Ciencias de la Comp., CACIC 2017, La Plata.

[37] Pinciroli F (2019) Modeling the Static View in Aspect-Oriented Software Development. Proceedings of the III International Congress on Computer Sciences and Information Systems, CICCSI 2019, Mendoza.