The origin and evolution of syntax errors in simple sequence flow models in BPMN

Regular paper

Joshua De Bock[1] and Jan Claes[1]

[1] Department of Business Informatics and Operations Management,
Ghent University, Tweekerkenstraat 2, 9000 Gent, Belgium
{joshua.debock, jan.claes}@ugent.be

Abstract. How do syntax errors emerge? What is the earliest moment that potential syntax errors can be detected? Which evolution do syntax errors go through during modeling? A provisional answer to these questions is formulated in this paper based on an investigation of a dataset containing the operational details of 126 modeling sessions. First, a list is composed of the different potential syntax errors. Second, a classification framework is built to categorize the errors according to their certainty and severity during modeling (i.e., in partial or complete models). Third, the origin and evolution of all syntax errors in the dataset are identified. This data is then used to collect a number of observations, which form a basis for future research.

Keywords: conceptual modeling, business process management, process, model, process of process modeling, quality, syntactic quality, syntax error

1 Introduction

Conceptual models are frequently used in practice and therefore it should come as no surprise that people are interested in the improvement of their quality [1, 2]. Therefore, we decided to study how quality issues arise and evolve during the modeling process. With this research we hope to provide a first insight into possible evolutions and the detection of syntax errors in early stages, as to improve the quality of process models. Because many factors influence the quality of conceptual models (e.g., the modeling goal, the domain of interest, the modeling language, the intended audience), this is a complex study domain and it was decided to limit the scope of the research in this initial phase. One of the oldest and most influential frameworks about the quality of conceptual modelling is the *SEQUAL* framework [2]. This framework makes a distinction between ***syntactic quality*** (symbol accordance with the modelling language syntax and vocabulary), ***semantic quality*** (correctness and completeness of the model in relation to reality), and ***pragmatic quality*** (understanding correctness of the model by its users).

Methodologically, it makes sense to first investigate syntactic quality. In contrast to for example semantic and pragmatic quality, syntactic quality can be measured more accurately because syntax errors can be detected and valued relatively more objectively [3]. Also, there is already a large body of knowledge related to syntactic quality. It appears to be included in most model quality frameworks (e.g., SEQUAL [3], CMQF [4]), reliable and valid metrics exist that measure syntactic quality (e.g., soundness of process models), and a high number of model editors contain features to prevent or detect syntax errors (e.g., Rational System Architect, ARIS). Although one may argue that it is less useful for practice to focus on syntax errors because tools help to avoid them, the practical value of this research lies exactly in the support for the development of such tools. The insights in the origin and evolution of syntax errors may bring forward the moment that tools can interact with the user about current or future syntax issues.

Next, mainly for practical reasons (i.e., the availability of a specific dataset), the scope of this paper is also reduced to only sequence flow process models, using a very limited subset of only 6 constructs from the popular BPMN language. The advantage is that the complexity of the research is reduced to its bare minimum. Obviously, this comes at the cost of a limited internal and external validity. Nevertheless, as you will be able to discover further in this paper, we still collected non-trivial observations that form a solid basis for future research.

The research was performed in three phases. First, based on the specification of the selected BPMN constructs, a comprehensive list was composed of potential syntax errors. Second, a classification framework was built that is used to categorize these potential errors according to their certainty and severity. Third, using the list and framework, the origin and evolution of the syntax errors that were made during a modeling session with 126 modelers was investigated in order to collect observations. As such, this paper describes 11 observations about the origin and evolution of syntax errors during modeling. They describe valuable insights, but they also illustrate the potential of the applied research method for future, more extensive, research.

This paper is structured as follows. Section 2 presents related work. Section 3 describes the construction of the list with syntax errors. Section 4 discusses the framework that can be used to classify syntax errors based on certainty and severity. Section 5 presents the collected observations. Section 6 provides a conclusion.

2 Related work

To the best of our knowledge, this is the first work to study the origin and evolution of syntax errors in conceptual models throughout the construction process. Nevertheless, this work builds further on studies about the quality of conceptual models and on research about conceptual modeling that takes a process orientation.

The prominent *SEQUAL* framework has been adapted and extended multiple times (e.g. by Krogstie, et al. [3], who make a distinction between 10 different types of

quality). A more recent effort, is the *Conceptual Modelling Quality Framework* (CMQF), which further extends the aforementioned frameworks [4]. As such, it synthesizes the above-mentioned *SEQUAL* extension and the *Bunge-Wand-Weber* (BWW) framework [5].

In order to put the study towards the origin and evolution of syntax errors into perspective, it can be considered in a stream of research that takes a process-oriented view on modeling. Hoppenbrouwers, et al. describe the main variables in what is called the ***process of conceptual modeling*** [6]. Wilmont, et al. add a cognitive level to this research and focus on individual differences as a key factor in the variation of errors between modeling efforts [7]. At the same time, Soffer, et al. lay the foundation for the study of the ***process of process modeling***, focusing on only one particular type of conceptual models (i.e., process models) [8]. This initiated a popular research stream about various aspects of the process of process modeling [9–11]. With insights in the origin and evolution of syntax errors, our research could improve the process of process modeling by assisting the modeler during the process.

3 Construction of the list of potential syntax errors

This section describes the creation of the list with potential syntax errors within the scope of the research (i.e., sequence flow models with a simplified BPMN syntax).

3.1 Approach

The BPMN 2.0 specification [12] was used to look up the definition and usage constraints of the sequence flow constructs of our tool. The six available constructs in the tool are (1) start event, (2) end event, (3) XOR (split or join) gateway, (4) AND (split or join) gateway, (5) activity, and (6) sequence flow. These are considered to be essential for sequence flow modeling and they were selected because they are most used in BPMN models [13]. Then, based on the specification, a list was built with the potential syntax errors (i.e., wrong usage of the symbols). Finally, the list was completed with other syntax issues that are similar to the real errors, but which are not wrong according to the syntax (cf. Section 4.3).

3.2 Results

Table 1 presents the composed list. It is an extension of the list by Claes, et al. [14]. The syntax issues that are not erroneous are marked in grey. From here on, we refer to ***syntax errors*** to indicate all issues in Table 1 that are not marked in grey and we use ***syntax issues*** to refer to all issues in the list (including the *syntax errors*).

Table 1. List of syntax issues with six constructs in BPMN 2.0

Construction	Code	
Contains no start event	0s	(0 start events)
Contains no end event	0e	(0 end events)
Contains multiple start events	S	(multiple starts)
Contains multiple end events	E	(multiple ends)
Sequence flow to start event	Bs	(between)
Sequence flow from end event	Be	(between)
Sequence flow from start event missing	Ms	(missing edges)
Sequence flow to end event missing	Me	(missing edges)
Not all of the paths are closed (missing end event?)	P	(path not closed)
Multiple parallel sequence flows from non-gateway	Sa	(missing AND split)
Multiple optional sequence flows from non-gateway	Sx	(missing XOR split)
Multiple parallel sequence flows towards non-gateway	Ja	(missing AND join)
Multiple optional sequence flows towards non-gateway	Jx	(missing XOR join)
Contains no gateways at all (but does contains multiple paths)	G	(no gateways)
No join gateways in case of optional iterations	I	(wrong iteration)
One gateway combines a join and split feature	C	(combination)
Wrong type of join combined with a certain split	W	(wrong type)
Gateway with only one incoming and one outgoing sequence flow	1	(1 edge in/out)
Wrong nesting of gateways	N	(wrong nesting)
AND and XOR are joined together in one join gateway	T	(joined together)
Infinite Loop	IL	(infinite loop)
Deadlock	DL	(deadlock)
Sequence flow between activities missing	Ma	(missing edges)
Sequence flow between gateways missing	Mg	(missing edges)
No label for activity	La	(missing label)
No label for edge departing from XOR splits	Lx	(missing label)

4 Construction and application of the classification framework

This section presents and discusses the classification framework that was built to categorize the *syntax issues* according to their certainty and severity.

4.1 Approach

Since we are interested in the evolution of errors (and related issues) during the modeling process, we were faced with the difficulty to recognize the *syntax issues* in an incomplete model. This is more challenging than one may expect at first sight. Let us illustrate this with an example. In sequence flow models, each element needs to be connected in the model in order to specify the order in which they should be considered (i.e., the sequence flow). In most modeling tools (including the one that was used for this research, cf. Section 5.1), a sequence flow arrow can be placed only between two existing components. Therefore, the modeler first has to create these two components and only then they can be connected with the arrow. But what if the connection of the two elements by placing the arrow is postponed? Since we do not know if this would be deliberate, it is (temporarily) hard to make a distinction between a planned delay and an actual *syntax issue*. Therefore, one dimension of the framework is the ***certainty*** of *syntax issues* in a partial or complete model.

Further, unfortunately, the specification of the BPMN 2.0 language [12] is not always completely consistent. For example, whereas it is explicitly stated that it is allowed to use a gateway that combines a join and a split function ("*a single Gateway could have multiple input and multiple output flows*", p. 90), it is not fully clear what the meaning is of this construction. The specification explains only the meaning of diverging and of converging gateways in detail. Furthermore, even when certain combinations of symbols are explicitly allowed and defined by the specification, because of the popularity of best practices and guidelines, modeling experts may still consider them to be wrong (e.g., omitting the AND split or XOR join gateway in certain cases). On the contrary, combinations also exist that are clearly not allowed according to the specification, but it is easy to guess what is meant (e.g., joining two *parallel* paths directly in an activity). These are often (mistakenly) considered to be correct. Therefore, the other dimension of the classification framework is the ***severity*** of *syntax issues* in a partial or complete model.

4.2 Certainty dimension

A distinction is made between ***wrong combinations of symbols*** and ***missing symbols*** during modeling. The former are *syntax issues* that can be resolved only by changing or removing something in the model, whereas the latter can be resolved by only adding symbols to the model. In case of wrong combinations of symbols, it is certain that a *syntax issue* exists. In the second case, the distinction between temporary planned incompleteness and unconsciously missing symbols cannot be made based on only the inspection of the partial model. Therefore, we introduce the notion of partial completeness to help assess the certainty of *syntax issues*. Every part in the model that is considered complete, will then by definition contain only ***definite*** *issues*. On the other hand, when a part of the model is considered incomplete, only the wrong combinations of symbols are considered *definite issues*, whereas missing symbols are considered ***uncertain*** *issues*.

We define ***completed parts*** of an incomplete sequence flow model as:

- the parts of the model between an opened split gateway that has been closed again by a join gateway, AND
- the parts of the model that are connected to an end event (in the direction of the sequence flows).

A number of remarks still need to be made. (1) When a model is sent to a model reader, it is considered to be complete and all parts are considered complete (even if the conditions for partial incompleteness are met). (2) This means that complete models cannot contain *uncertain issues*. Every *syntax issue* in a complete model is a *definite issue*. (3) All *uncertain issues* will thus eventually evolve into *definite issues* unless the modeler adds the missing symbols or changes the erroneous construction.

4.3 Severity dimension

Since there can be a discussion whether a *syntax issue* is a real error in certain cases, we also make a distinction between different severity levels. We define three severities of *syntax issues*.

- First, an ***error*** is when the *syntax issue* is clearly wrong according to the specification.
  Are considered an error: 0s, 0e, Bs, Be, Ms, Me, Ma, Mg, P, Ja, 1, W, T, IL.
- Second, an ***irresolution*** is when the specification is not completely clear or when it is inconsistent.
  Are considered an irresolution: D, G, I, C, N, La, DL.
- Third, a ***confusion*** is when the *syntax issue* is clearly correct according to the specification, but nevertheless it is widely considered a bad practice because it hinders the (ease of) understanding.
  Are considered confusing: Jx, Sx, S, E, Sa, Lx.

4.4 Transformations

The two levels of certainty – uncertain (U) and definite (D) – and the three levels of severity – error (E), irresolution (I), and confusion (C) – provide six combinations: uncertain error (UE), uncertain irresolution (UI), uncertain confusion (UC), definite error (DE), definite irresolution (DI), and definite confusion (DC). Not every transformation between these types is possible. The *uncertain* types can evolve into *definite* types (e.g., when the part is completed without correcting the *issue*) or they can be resolved by the modeler. They cannot transform (directly) into another *uncertain* type. On the other hand, *definite* types can transform into other *definite* types or they can be resolved. They cannot transform (again) into an *uncertain* type. Table 2 presents an overview. Possible transformations are marked with an "X", and "/" refers to 'no issue'.

Table 2. Possible transformations between the types of syntax issues, marked with an "X"

From \ To	UE	UI	UC	DE	DI	DC	/
UE	-			X	X	X	X
UI		-		X	X	X	X
UC			-	X	X	X	X
DE				-	X	X	X
DI				X	-	X	X
DC				X	X	-	X
/	X	X	X	X	X	X	-

5 Investigation of the origin and evolution of syntax issues during modeling

This section discusses how the list and classification framework were used to analyze the origin and evolution of *syntax issues* during modeling.

5.1 Approach

For this research, an existing data set was used (the same as by Claes, et al. [14]). It contains the data of a modeling experiment in which the participants were instructed to construct a sequence flow model based on a textual description of a process (at a certain point in the experiment task flow, cf. [14]). The participants were 126 master students of Business Engineering at Ghent University who were enrolled in a Business Process Management course in which they learned the BPMN 2.0 syntax and how to create models within this language. The tool used to collect the data, is the Cheetah Experimental Platform[1]. It contains a simplified BPMN modeling editor offering the six constructs described above (cf. Section 3.1). It was selected for its features to log every operation of the user in an event log and to replay the modeling afterwards. This latter feature was used to evaluate after each operation of the user whether a *syntax issue* arose, and which was the kind and type of the *syntax issue*. For each of the 126 modeling instances, we thus complemented the dataset with the timing, kind, and type of *syntax issues* during modeling. This allowed performing a number of interesting analyses, which are discussed below.

5.2 Syntax issues during and after modeling

First, it was examined which types of *syntax issues* were made during modeling. On average, each sequence flow model contained 2.4 UEs, 1.0 UIs, 4.3 UCs, 3.2 DEs, 2.3 DIs, and 5.5 DCs during modeling. Since certain of these *issues* can evolve into others, this does not mean that each model contained on average 18,7 different *syntax issues* during modeling (the sum of the aforementioned numbers). After modeling, there are on average 0.5 DEs, 2.2 DIs, and 3.6 DCs.

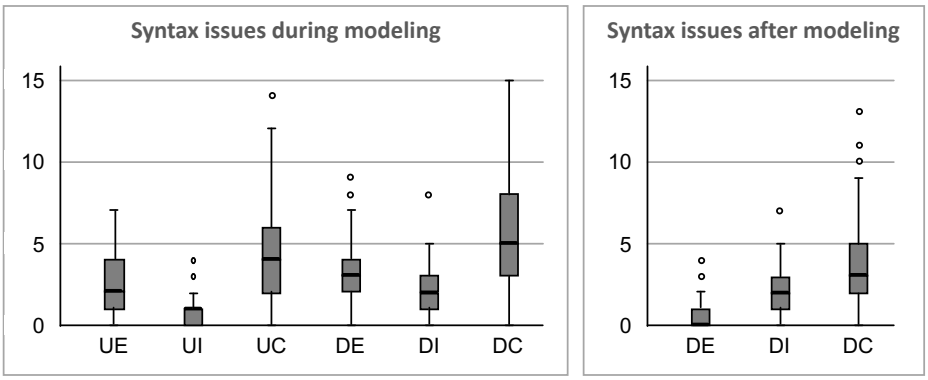

Fig. 1. Boxplots of the number of syntax issues per model during and after modeling

[1] Download and info at http://bpm.q-e.at/?page_id=56 (dd. 16/03/0218)

Fig. 1 shows more details on the spread of *syntax issues* during and after modeling. Based on Fig. 1, a number of observations can be made:

Obs1. The minimum occurrence of each *syntax issue*, both during and after modeling, is 0. The dataset confirms that 2 of the 126 (1.5%) did not have any *syntax issue* during the whole modeling process.

Obs2. Even when ignoring the outliers, the variance of the occurrence of *syntax issues* is relatively high (0 to ≥5 for most types).

Obs3. Confusions (UC and DC) occur more than the other types of *syntax issues*.

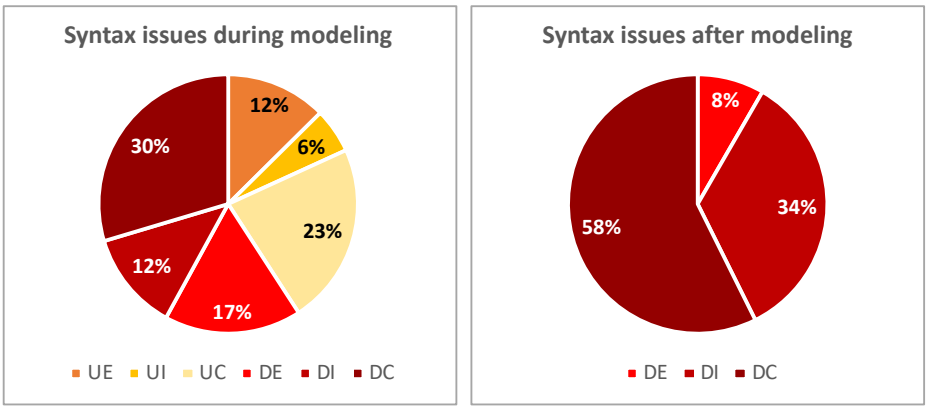

Fig. 2. Types of syntax issues during and after modeling

Next, the *relative* occurrence of each type of *syntax issue* during and after modeling is presented in Fig. 2.

Obs4. Also in Fig. 2 it can be noticed that confusions occur more than the other types of *syntax issues* (cf. Obs3).

5.3 The origin of syntax issues during modeling

In order to investigate the origin of a *definite syntax issue*, we examined what happened at the time of the operation that caused the *issue*. When another type of *issue* disappeared with that same operation, the operation is considered to have transformed one type into another type (with the restriction that no *issue* can evolve into an(other) *uncertain issue*). If no other *issue* type disappeared at the same time, the *definite syntax issue* was considered to be initiated at that exact point in time (denoted with origin "/").

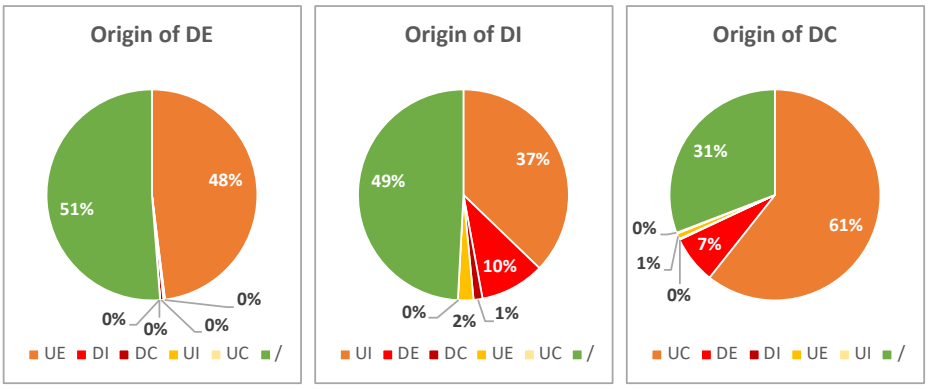

Fig. 3. The origin of syntax issues during modeling

Fig. 3 shows an overview of the origins of the *definite syntax issues* during modeling. Again, a number of interesting observations can be made:

Obs5. A *definite issue* often has its origin in an *uncertain issue* of the same severity (orange slices). In this case, the *issue* could thus already be detected in an incomplete part of the partial model.

Obs6. In the other cases, they mostly are created directly (green slices). Only rarely they originate from another type of *definite issue* (red slides, ≤10%) or from another type of *uncertain issue* (yellow slices, ≤2%).

5.4 The evolution of syntax issues during modeling

Based on the first of the previous set of observations (i.e., Obs5), one may wonder if an *uncertain* type of *syntax issue* always evolves into a *definite* type of the same severity. Therefore, it is interesting to see in which other types the *uncertain* types evolve during modeling, which is represented in Fig. 4.

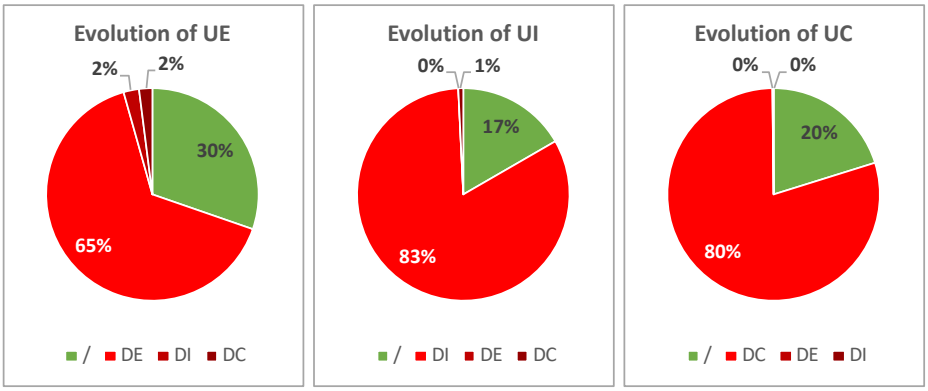

Fig. 4. The evolution of uncertain syntax issues during modeling

These are the observations related to Fig. 4:

Obs7. In the majority of cases (red slices, ≥65%) the *uncertain syntax issue* was transformed later on in the corresponding *definite* type of *issue*. This means that the *syntax issue* can indeed already be detected in an incomplete part of the partial model (cf. Obs5).

Obs8. In a smaller number of cases (green slices, 17-30%), the *issue* was resolved before the model part was completed (because then it would be transformed into a *definite issue*, which are the red slices). Potentially, they were never a real *issue*, but rather the manifestation of the postponement of actions, which introduced temporary *syntax issues*.

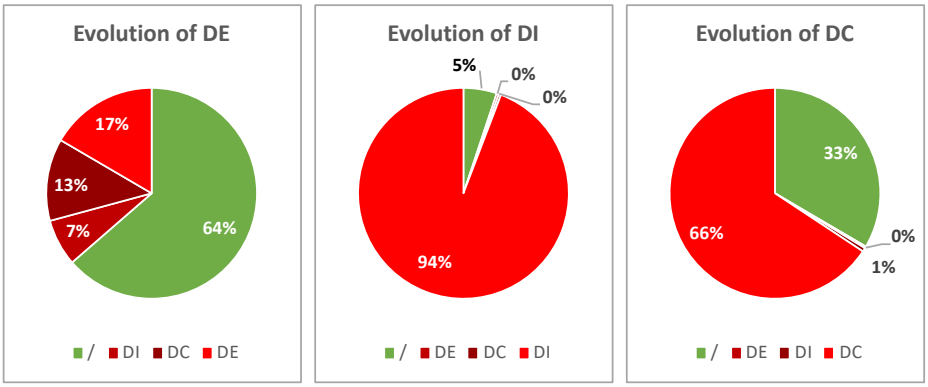

Fig. 5. The evolution of definite syntax issues during modeling

Further, it is also interesting to see what happens with the *definite syntax issues* during modeling. Fig. 5 shows in what other types of *issues* they evolved. It can be observed:

Obs9. Most (64%) of the *definite errors* (DE) are resolved before completing the model. Nevertheless 17% of these errors remain in the final model. Some others turn into an irresolution (7%) or a confusion (13%).

Obs10. In contrast, the majority (94%) of *definite irresolutions* (DI) are never resolved. Remember that 37% of all DIs could already be detected in an incomplete part of the partial model (cf. Obs5 and Fig. 3).

Obs11. Similarly, 66% of de *definite confusions* (DC) remain in the final model, whereas 61% of all DCs can be detected early (cf. Obs5 and Fig. 3).

6 Conclusion

Based on the specification of the BPMN 2.0 syntax for process modeling, we derived a list of potential syntax errors and related issues that can occur in simple sequence flow models. As already proposed by Natschläger, et al., the definitions of constructs are scattered over different pages of the BPMN specification and they are not always unambiguously defined [15]. Therefore, a classification framework was

constructed to categorize the issues in the list according to the certainty (uncertain or definite) and severity (error, irresolution, confusion). Further, we analyzed the data of 126 modeling instances and added the timing, kind (i.e., according to the list), and type (i.e., according to the framework) of each detected issue to the data.

The results are provisional (cf. limitations below), but interesting. Most (64%) of the real syntax errors (DE) that were made during the modeling session from the dataset were corrected. They were not present anymore in the final model. Only 17% remained (the other 19% were transformed in other issues). Moreover, 48% of all the real errors (DE) during modeling could be detected at an early stage, when the part of the model in which they occurred was still not completed.

Further, except for real errors (DE), we also collected information about irresolutions (DI), which are syntax issues for which experts would not agree if they are actually correct or not (for example a single gateway combining a join and split function). Irresolutions (DI) were seldom corrected (only 5%). Interestingly, 37% of them could be detected at an early stage. Similarly, confusions (DC) are constructions that are definitely correct, but that should advisably be avoided (such as certain occasions of multiple arrows arriving or originating in an activity without using an explicit gateway). Not less than 66% of them are never removed after originating. Yet, 61% can be detected early. An average model from the dataset contained 3.2 errors (DE), 2.3 irresolutions (DI), and 5.5 confusions (DC) during modeling (of which on average 0.5 DE, 2.2 DI, and 3.6 DC remained in the final model).

These conclusions indicate that it could be useful to study the origin and evolution of syntax issues in more detail. This can advance the tool features that aim to detect and prevent syntax issues. It may also produce interesting knowledge for modeling teachers, because in a learning context it is always better to focus on the root cause of problems. Therefore, we propose that future research focuses on dealing with the limitations of this study on the one hand and on extending the scope of the research on the other hand.

Being an explorative study that aims to reveal initial insights in the origin and evolution of syntax issues in conceptual models, this study has a number of limitations. First, the dataset is an arbitrary dataset, which is definitely not representative for all modelers (for example it contains only data of student observations). Next, the used list and the used framework are not evaluated. There is a real chance that they are not complete. Further, the analysis was performed by a limited number of people. Since the coding of syntax issues was very labor-intensive, the probability of mistakes is real. On the other hand, the dataset is considered big enough to warrant a certain degree of reliability of the results.

Finally, we plan to extend the study in several ways. Whereas the current analysis is limited to the evolution of the three generic severity types, in future work the evolution analysis will focus on all the different kinds of issues in the list. Future work may also include an extension towards other (process) modeling languages (including the full set of BPMN constructs) and towards other types of quality.

References

1. Rockwell, S., Bajaj, A.: COGEVAL: Applying cognitive theories to evaluate conceptual models. Adv. Top. Database Res. 4, 255–282 (2005).
2. Lindland, O.I., Sindre, G., Solvberg, A.: Understanding quality in conceptual modeling. IEEE Softw. 11, 42–49 (1994).
3. Krogstie, J., Sindre, G., Jørgensen, H.: Process models representing knowledge for action: A revised quality framework. Eur. J. Inf. Syst. 15, 91–102 (2006).
4. Nelson, H.J., Poels, G., Genero, M., et al.: A conceptual modeling quality framework. Softw. Qual. J. 20, 201–228 (2012).
5. Wand, Y., Weber, R.: An ontological model of an information system. IEEE Trans. Softw. Eng. 16, 1282–1292 (1990).
6. Hoppenbrouwers, S.J.B.A., Proper, H.A., Van der Weide, T.P.: A fundamental view on the process of conceptual modeling. In: Delcambre, L. et al. (eds.) Proc. ER '05. LNCS 3716. pp. 128–143. Springer (2005).
7. Wilmont, I., Hengeveld, S., Barendsen, E., et al.: Cognitive Mechanisms of Conceptual Modelling: How Do People Do It? In: Ng, W. et al. (eds.) Proc. ER '13. LNCS 8217. pp. 74–87. Springer (2013).
8. Soffer, P., Kaner, M., Wand, Y.: Towards understanding the process of process modeling: Theoretical and empirical considerations. In: Daniel, F. et al. (eds.) Proc. BPM '11 Workshops. LNBIP 99. pp. 357–369. Springer, Clermont-Ferrand, France (2012).
9. Pinggera, J., Soffer, P., Fahland, D., et al.: Styles in business process modeling: An exploration and a model. Softw. Syst. Model. 14, 1055–1080 (2013).
10. Claes, J., Vanderfeesten, I., Pinggera, J., et al.: A visual analysis of the process of process modeling. Inf. Syst. E-bus. Manag. 13, 147–190 (2015).
11. Claes, J., Vanderfeesten, I., Gailly, F., et al.: The Structured Process Modeling Theory (SPMT) - A cognitive view on why and how modelers benefit from structuring the process of process modeling. Inf. Syst. Front. 17, 1401–1425 (2015).
12. OMG: Business Process Model and Notation (BPMN) version 2.0. (2011).
13. Zur Muehlen, M., Recker, J.: How much language is enough? Theoretical and practical use of the Business Process Modeling Notation. In: Bellahsène, Z. and Léonard, M. (eds.) 20th International Conference on Advanced Information Systems Engineering, CAiSE 2008, Proceedings. LNCS 5074. pp. 465–479. Springer, Montpellier, France (2008).
14. Claes, J., Vanderfeesten, I., Gailly, F., et al.: The Structured Process Modeling Method (SPMM) - What is the best way for me to construct a process model? Decis. Support Syst. 100, 57–76 (2017).
15. Natschläger, C.: Towards a BPMN 2.0 ontology. Lect. Notes Bus. Inf. Process. 95 LNBIP, 1–15 (2011).