Implementation of understandability metrics for BPMN models using R

Jonas Lieben

July 3, 2018

1 Implementation details of metrics

All metrics are implemented in R and can be calculated using the understandBPMN package. This package makes use of the tidyverse package which includes dplyr, purrr and ggplot2 among others. Another used packages is XML. XML is used because the BPMN-files are using the XML-standard. It makes querying the files easy and supports the use of XPath-expressions.

For each of these metrics, the definition, the related work, the related aspects, the function name, the interpretation, the range and an example are provided. The definition explains how the metric should be calculated for BPMN processes. The related work contains the references to scientific articles and papers in which the metric appeared. The related aspects are the concepts which are linked to the metric. The function name is the name of the corresponding function in the R package. The interpretation contains the effect of an increase in the metric on the overall understandability of a process model. The range describes the minimum and maximum value the metric can have. Lastly, the example describes how the metric is calculated for an example BPMN model.

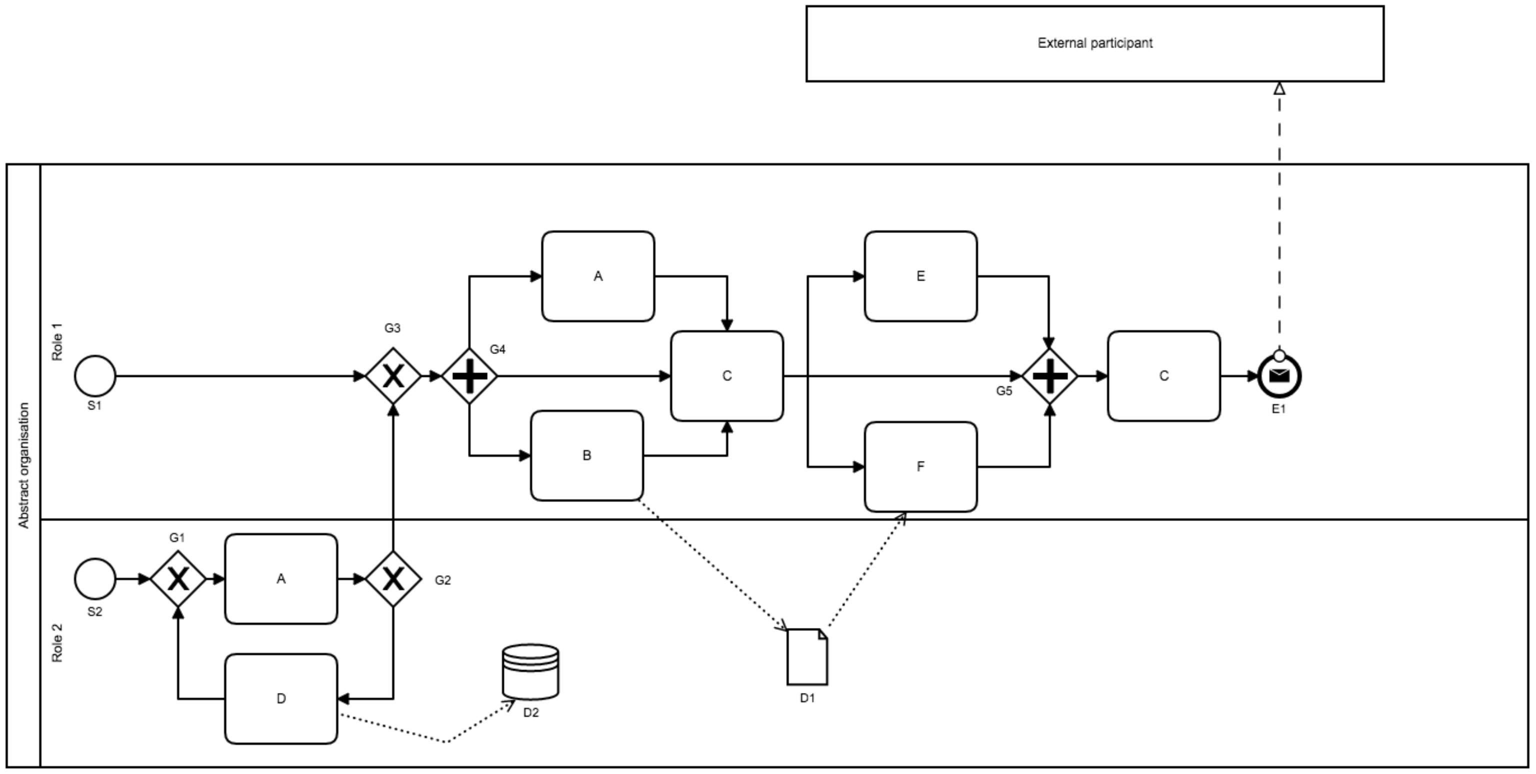

Figure 1: Example process model to calculate metrics

To illustrate the calculation of metrics, an example is used. This example can be found in figure 1. This example includes:

- 8 activities: objects with the label A, D, A, B, C, E, F and C.
- 5 gateways: objects with the label G1, G2, G3, G4 and G5.
- 2 data objects: objects with the label D1 and D2.
- 3 events: objects with the label S1, S2 and E1
- 20 sequence flows: arrows with a solid line
- 1 message flow: arrow with a dashed line connecting E1 with the external participant
- 2 swimlanes: objects with the label role 1 and role 2
- 2 pools: objects with the label abstract organisation and role external participant

There are a two particularities which need to be highlighted in this model. Firstly, the first activity C is not joined with a parallel gateway. The consequence is that the incoming flows to this activity are handled as if there is an exclusive join gateway before activity C. This means that the first activity C and everything after it is executed three times. Secondly, there is no gateway after the first activity C. When an activity has multiple outgoing sequence flows, this is handled as a parallel gateway. In other words, both E and F are executed after C. This makes the model complex. This complexity is added to the model in order to demonstrate the workings of metrics such as structuredness.

1.1 Process model size

Definition: Process model size is defined as the number of nodes. The translation to the BPMN notation is the sum of the number of activities, gateways, events and data objects.

Related work: [8, 12, 9]

Related aspects: Activities, events and data objects, gateways

Function name R package: size_process_model(file_path)

Interpretation: The bigger the size, the less understandable the model.

Range: $[0, +\infty[$

Example: The size in the example is equal to the sum of eight activities, five gateways, two data objects and three events, which equals 18. In conclusion, the process model size equals 18 in the example.

1.2 Number of empty sequence flows

Definition: An empty sequence flow connects a parallel split gateway with a parallel join gateway without any activity in between. The sequence flow is redundant and clutters the model. The same situation appears when an activity with multiple outgoing sequence flows is directly connected to a parallel join gateway. In other words, it is semantically the same to use a parallel split gateway or to use an activity with multiple outgoing sequence flows for BPMN models.

Related work: [4]

Related aspects: Flows

Function name R package: n_empty_sequence_flows(file_path)

Interpretation: The more empty sequence flows, the less understandable the model.

Range: $[0, +\infty[$

Example: There is one empty sequence flow present in the example. This is the flow connecting activity C with gateway G5. Notice that the sequence flow connecting G4 and C is not an empty sequence flow, because a parallel join gateway is not used to join the sequence flows. Instead, activity (C) joins multiple incoming sequence flows.

1.3 Number of duplicate tasks

Definition: A duplicate task is a task which has the same label as another task in the diagram. This assumes that each task has exactly one label and that a label can be used in multiple tasks. The number of duplicate tasks is equal to the number of tasks carrying the same label minus one summed for each label. We define it as follows:

$$\#\, duplicate\ tasks = \sum\nolimits_{l \in L} \#\, tasks\ with\ label\ l - 1$$

Related work: [6]

Related aspects: Activities, events and data objects

Function name R package: n_duplicate_tasks(file_path)

Interpretation: The more duplicate tasks, the more understandable the model becomes to a certain extent. However, it is important to note that the process model size increases as well. A bigger process model size decreases the understandability. Therefore, there is not a monotonic relation between the understandability and the number of duplicate tasks.

Range: $[0, +\infty[$

Example: There are multiple duplicate tasks present in this model. There are two tasks carrying the label A and two tasks carrying the label C. The number of duplicate tasks is therefore 1 + 1 = 2.

1.4 Number of pools

Definition: Within the context of BPMN, a pool is an artifact in which all other elements such as activities, events, gateways and sequence flows are placed. A pool represents an organisation. This can be either the organisation that hosts the process or an external participant such as a customer or supplier.

Related work: [13]

Related aspects: Pools and swimlanes

Function name R package: n_pools(file_path)

Interpretation: The more pools, the less understandable the model.

Range: $[0, +\infty[$

Example: The number of pools is equal to two. There is one pool representing an external participant and one pool with the label abstract organisation.

1.5 Number of swimlanes

Definition: Within the context of BPMN, a swimlane is a part of the pool and represents a certain role, function or person in the organisation.

Related work: [5]

Related aspects: Pools and swimlanes

Function name R package: n_swimlanes(file_path)

Interpretation: The more swimlanes, the less understandable the model.

Range: $[0, +\infty[$

Example: There are two swimlanes in the model: one with the label 'Role 1' and one with the label 'Role 2'.

1.6 Number of message flows

Definition: A message flow is an arrow which links two pools. The arrow represents communication between two pools.

Related work: [13]

Related aspects: Flows

Function name R package: n_message_flows(file_path)

Interpretation: The more message flows, the less understandable the model.

Range: $[0, +\infty[$

Example: There is one message flow in the example connecting E1 with the external participant.

1.7 Number of data objects

Definition: A data object can either be a document or a data store.

Related work: [13]

Related aspects: Activities, events and data objects

Function name R package: n_data_objects(file_path)

Interpretation: The more documents and data stores, the less understandable the model.

Range: $[0, +\infty[$

Example: There are two data objects in the example: a data store with the label D2 and a document with the label D1.

1.8 Density

Definition: The density indicates how connected the nodes are in the diagram. It is defined as the number of sequence flows, divided by the theoretical maximum number of sequence flows:

$$\frac{\#\,sequence\,flows}{(process\,model\,size\ -\ \#\,data\,objects)\times\ (process\,model\,size\ -\ \#\,data\,objects\ -\ 1)}$$

Related work: [9, 14]

Related aspects: Activities, events and data objects, flows, gateways

Function name R package: density_process_model(file_path)

Interpretation: The higher the density, the less understandable the model.

Range: $[0, 1]$

Example: The number of sequence flows equals 20 in the example. The process model size equals 18. The density is therefore equal to $\frac{20}{(18-2)\times(18-2-1)} =$ 0.08.

1.9 Coefficient of network connectivity

Definition: The coefficient of network connectivity is the average number of sequence flows per object in the model and is defined as:

$$\frac{\#\,sequenceflows}{\#\,process\ model\ size\ -\ \#\,data\ objects}$$

Related work: [2, 9, 14]

Related aspects: Activities, events and data objects, gateways, flows

Function name R package: coefficient_network_connectivity(file_path)

Interpretation: The higher the coefficient of network connectivity, the less understandable the model.

Range: $[0, process\ model\ size -\ \#\ data\ objects\ - 1]$

Example: The number of sequence flows equals 20 in the example. The size equals 18. The coefficient of network connectivity is therefore equal to $\frac{20}{18-2} = 1.25$.

1.10 Connectivity level between pools

Definition: The connectivity level between pools is defined as:

$$\frac{\#\,message\ flows}{\#\,pools}$$

Related work: [2]

Related aspects: Pools and swimlanes, flows

Function name R package: connectivity_level_between_pools(file_path)

Interpretation: The higher the connectivity level between pools, the less understandable the model.

Range: $[0, +\infty[$

Example: The connectivity level between pools in the example is equal to $\frac{1}{2} = 0.5$.

1.11 Average connector degree

Definition: A connector is a gateway or an activity with multiple incoming or outgoing sequence flows in BPMN notation. The average connector degree is a metric which indicates how many incoming and outgoing sequence flows all connectors have. We define a connector activity as an activity which has at least two incoming or two outgoing sequence flows. In order to aggregate all information concerning incoming and outgoing sequence flows into one metric, the average is used. It is defined as

$$\frac{\sum_{all\ gateways\ and\ connector\ activities} incoming\ sequence\ flows + \ outgoing\ sequence\ flows}{\#\ gateways + \#\ connector\ activities}$$

If there are no gateways or connector activities present in the model, zero is returned.

Related work: [9, 14]

Related aspects: Flows, gateways

Function name R package: avg_connector_degree(file_path)

Interpretation: The higher the average connector degree, the less understandable the model.

Range: $[3, +\infty[$ (if we make abstraction of the case in which there are no connectors)

Example: In this example, the first activity C is a connector activity. This activity has 6 linked sequence flows. G1, G2 and G3 each have 3 linked sequence flows. G4 and G5 have 4 linked sequence flows. The sum of all those sequence flows equals 23. The number of gateways equals 5 and there is one connector activity. The average connector degree equals $\frac{6+3+3+3+4+4}{5+1} = 3.83$

1.12 Maximum connector degree

Definition: The definition of connector and connector activity of section 1.11 is used. The maximum connector degree is a metric which determines how many incoming and outgoing sequence flows the connector with the most incoming and outgoing sequence flows has. It is defined as

$$\forall \, gateways \; and \; connector \; activities$$

$$max(incoming + outgoing \; sequence \; flows)$$

If there are no gateways or connector activities present in the model, zero is returned.

Related work: [9, 14]

Related aspects: Flows, gateways

Function name R package: max_connector_degree(file_path)

Interpretation: The higher the maximum connector degree, the less understandable the model.

Range: $[3, +\infty[$ (if we make abstraction of the case in which there are no connectors)

Example: The connector with the most incoming and outgoing sequence flows is the first activity C. The number of linked sequence flows is equal to six, which is in this example the maximum connector degree.

1.13 Sequentiality

Definition: The sequentiality metric expresses how sequential the process is modelled. A process is completely sequential when no connectors are used. Consequently, there is no room for concurrency, choice or loops. To calculate sequentiality, non-connector activities should be used. We define non-connector activities as activities with exactly one incoming and one outgoing sequence flow. We define a sequential sequence flow as a sequence flow connecting non-connector activities or a sequence flow connecting an event with a non-connector activity. This event should have not more than one incoming and one outgoing sequence flow. Sequentiality is defined as

$$\frac{\# \, sequential \; sequence \; flows}{\# \, sequence \; flows}$$

Related work: [2, 3, 9, 14]

Related aspects: Activities, events and data objects, gateways, flows, partitionability

Function name R package: sequentiality(file_path)

Interpretation: The higher the sequentiality, the more understandable the model.

Range: $[0, 1]$

Example: The number of sequential sequence flows in the example is one. This is the sequence flow connecting activity C with E1. The number of all sequence flows equals 20. Sequentiality is hence $\frac{1}{20} = 0.05$

1.14 Cyclicity

Definition: Cyclicity indicates the degree that it is possible to repeat some activities or sequences of activities. It is defined as

$$\frac{\#\ nodes\ on\ cycle}{process\ model\ size - \#\ data\ objects}$$

A cycle or loop is defined as a group of objects (ie activities, gateways and events) which are modelled in such a way that it is possible to repeat them. In order to model these loops, at least one split activity or gateway and one join activity or gateway are needed.

Related work: [9, 10]

Related aspects: Loops, activities, events and data objects, gateways

Function name R package: cyclicity(file_path)

Interpretation: The higher the cyclicity, the less understandable the model.

Range: $[0, 1]$

Example: To calculate the cyclicity, a path log is created. This path log contains all possible paths in the model. We define a path in a similar way as a path in graph theory and not as a trace. This means that each path starts with a start event and ends with an end event and that if there is a parallel or inclusive split gateway, only one of the outgoing sequence flows is chosen and becomes part of the path. This is in contrast with a trace which contains all objects connected with the outgoing sequence flows of a parallel split gateway. The implication of our definition of a path is that the behaviour of all connectors is considered as an exclusive gateway. This assumption is made in order to make the calculation of several metrics easier and because this definition of paths is used in several metrics such as the diameter.

During the creation of the path log, all loops are identified. This identification allows to calculate the number of nodes on a cycle. We will first introduce the algorithm to create the path log, then we will illustrate the creation of one part of a path with our example and lastly, we will give examples of a couple of paths. Notice that the identification of loops in a path is done in a separate function, searchLoops, in order to make the algorithm easier to understand. searchLoops is also explained using an example.

In the algorithm the following concepts are used:

- id: the id of an object
- target: the object to which the sequence flow is directed
- source: the object from which the sequence flow starts

```
    Result: Path log and repetitions
 1  sequence_flows ← all sequence flows with their corresponding ids,
      targets and sources;
 2  start_events ← all start events;
 3  path_log ← start_events ;
 4  repetitions ← array(0) ;
 5  i ← 1;
 6  while i <= length(path_log) do
 7      repeat
 8          current_artifact ← last_element(path_log[i]);
 9          number_in_flows_current_artifact ←
              length(distinct(sequence_flows(target ==
              current_artifact)));
10          number_out_flows_current_artifact ←
              length(distinct(sequence_flows(source ==
              current_artifact)));
11          join_before_artifact ← number_in_flows_current_artifact >= 2;
12          split_after_artifact ← number_out_flows_current_artifact >= 2;
13          next_artifacts ←
              sequence_flows(source == current_artifact).target;
14          if join_after_artifact then
15              path_log[i].add("join")
16          end
17          if length(next_artifacts) ! = 0 then
18              if split_after_artifact then
19                  first_part_path ← path_log[i];
20                  search_loops ← searchLoops(first_part_path);
21                  if search_loops.first_element_after_split != "no loop
                      found" then
22                      next_artifacts. filter(target !=
                          search_loops.first_element_after_split_in_loop);
23                      repetitions.add(search_loops.repetition_from_loop);
24                  end
25                  for j in 1 till length(next_artifacts) do
26                      if j == 1 then
27                          path_log[i].add("split", next_artifacts[j]);
28                      else
29                          path_log.add(first_part_path, "split",
                              next_artifacts[j]));
30                      end
31                  end
32              else
33                  path_log[i].add(next_artifacts[1]);
34              end
35          else
36              break ;
37          end
38      until ;
39      i ← i + 1 ;
40  end
41  return (path_log, repetitions);
```

Algorithm 1: Creates a path log and a list of repetitions

First, all sequence flows are saved with an id, their target and source (line 1). This can be an example object of a sequence flow: sequence_flow(id = "sf__s1_G3", target = "G3", source = "S1"). All these objects are saved in an array. Next, all start events are saved in start_events (line 2). In our example, these are S1 and S2. They are copied into the variable path_log (line 3). Afterwards, repetitions is initialised and one is assigned to i (line 4 and 5).

Then, two loops are started. The while loop, which iterates over all paths, and the repeat loop, which iterates until an end event is reached. Let's assume that we start with S2. There is no split or join after the artifact S2 since it has zero incoming flows and one outgoing flow (line 9-12). Next_artifacts contains one object, gateway G1 (line 13). Therefore, the length of next_artifacts is not equal to 0 (line 17). There is no split after S2, which means that G1 is added to the path (line 33). Again, the current artifact is identified (line 8). In this case, it is G1. G1 has two incoming sequence flows and one outgoing sequence flow (line 9 and 10). Next, a join is detected (line 14). "join" is added to the path (line 15) and there are no splits. A is added to the path (line 33) and becomes the current artifact (line 8). In the following iteration G2 is added to the path.

When G2 is the current artifact (line 8), a split is detected, because G2 has two outgoing sequence flows (line 12). The first part of the path, ie ¡S2, join, G1, A, G2¿ is saved in the corresponding variable (line 19). A loop check is performed (line 20). As there is no repetition detected in the path, the function searchLoops returns an object with the attribute first_element_after_split equal to "no loop found". Then "split" and D are added to the current path (line 27). In addition, a new path is created which equals ¡S2, join, G1, A, G2, split, G3¿ (line 29). We continue, however with the previous path for further illustration of the example. G1, A and G2 are added after D. When G2 is again the current artifact, the function searchLoops will in this case return an object with the attribute first_element_after_split equal to D (line 20). Remember that the path is currently ¡S2, join, G1, A, G2, split, D, join, G1, A, G2¿. Notice that there is a repetition in the path, which is detected by the function searchLoops (line 20). The activity D is removed from next_artifacts in order to prevent the algorithm from infinitely running (line 22). Also the repetition join, G1, A, G2, split, D is saved in the array repetitions (line 23). Next the elements split and G3 are added to the path (line 27). This algorithm continues until for all paths in the path_log E1 is reached. Notice that during the creation of one path, more paths may be added to the log when a split appears.

Examples paths of the resulting path log are:

- <S2, G1, join, A, G2, split, D, G1, join, A, G2, split, G3, join, G4, split, A, C, join, split, E, G5, join, C, E1>

- <S2, G1, join, A, G2, split, D, G1, join, A, G2, split, G3, join, G4, split, C, join, split, E, G5, join, C, E1>
- <S2, G1, join, A, G2, split, G3, join, G4, split, B, C, join, split, E, G5, join, C, E1>
- <S1, G3, join, G4, split, A, C, join, split, E, G5, join, C, E1>

The total number of paths in the path log is 27.

The algorithm to detect loops is described in algorithm 2.

```
   Result: Find loops in path
 1 indices_split ← which(path.element == ”split”);
 2 indices_join ← which(path.element == ”join”);
 3 index_join_2 ← indices_join[length(indices_join)];
 4 index_split_1 ← indices_split[length(indices_split[indices_split¡
     index_join_2])];
 5 index_second_join ← indices_join[2];
 6 index_first_split ← indices_split[1];
 7 loop_found ← false ;
 8 while index_join_2 ⩾ index_second_join && !loop_found do
 9 |   while index_split_1 ⩾ index_first_split && loop_found do
10 |   |   loop_length ← length(path) - index_join_2;
11 |   |   index_join_1 ← index_split_1 - 1 - loop_length;
12 |   |   if index_join_1 ⩾ 1 then
13 |   |   |   sub_path_1 ← path[index_join_1 till index_split_1];
14 |   |   |   sub_path_2 ← path[index_join_2 till length(path)];
15 |   |   |   if length(sub_path_1) > 2 then
16 |   |   |   |   loop_found ← all_elements_equal(sub_path_1,
   |   |   |   |     sub_path_2);
17 |   |   |   |   if loop_found then
18 |   |   |   |   |   repetition ← path[index_join_1 till index_join_2];
19 |   |   |   |   |   first_element ← path[index_split_1 + 1];
20 |   |   |   |   end
21 |   |   |   end
22 |   |   end
23 |   |   index_split_1 ← index of split before split_1;
24 |   end
25 |   index_join_2 ← index of join before join_2;
26 |   index_split_1 ← index of split before join_2;
27 end
28 if loop_found then
29 |   return (first_element, repetition);
30 else
31 |   return ”no loop found”;
32 end
```

Algorithm 2: Search for loops in path

Suppose we use again the path <S2, G1, join, A, G2, split, D, G1, join, A, G2> as an example to illustrate the workings of the function. Indices_split contains the array [6] (line 1) and Indices_join contains the array [3, 9] (line 2). Index_join_2 contains the index of the last join, ie 9 (line 3). Index_split_1 is the index of the split just before the last join, which is in this case 6 (line 4). Index_second_join is equal to the index of the second join in the path, which is again 9 (line 5). Index_first_split is the index of the first split, which equals 6 (line 6). Afterwards, loop_found is set to false. (line 7) As long as index_join_2 is bigger than or equal to 9 and index_split_1 is bigger than or equal to 6 and no loop is found (line 8 and 9), the loop length is calculated by subtracting index_join_2 from the length of the path. In this case it is equal to 11 - 9 = 2 (line 10). index_join_1 is expected to be at index_split_1 - 1 - loop_length. In this case index_join_1 equals to 6 - 1 - 2 = 3 (line 11). sub_path_1 equals to the path from index_join_1 till index_split_1, which is <join, A, G2> (line 13). sub_path_2 equals to the path from index_join_2 till the end of the path, which is <join, A, G2> (line 14). All elements in the sub paths equal each other (line 16). Therefore, a loop is found, the variable repetition is assigned <join, A, G2, split, D, G1> (line 18) and the first_element variable contains D (line 19). A loop is found and therefore, the object (D, <join, A, G2, split, D, G1>) is returned (line 29).

If there are more splits and joins, iterations are performed over the entire path in order that all possible combinations of pairs of joins and splits are found. These iterations continue until a loop is found or until the first split and the second join is reached for split_index_1 and join_index_2. Split_index_2 is never used, because the index of the last split is the same as the length of the trace. The algorithm to search for loops is made this complex, because otherwise it is not possible to detect nested loops. Nested loops occur when a loop is a part of a bigger loop. These should be detected as well in order that algorithm 1 will not get stuck in an infinite loop. The function searchLoops to detect repetitions makes algorithm 1 less scalable. The calculation time increases exponentially when a loop is combined with a choice or concurrent construct having many possible options or concurrent activities.

During the creation of the path log, all repetitions are saved. The elements in the repetitions are the nodes on a cycle. In the example, these elements are join, A, G2, split, D and G1. Split and join are removed and the number of elements is calculated. In this case, it equals to 4. The process model size equals 18. The cyclicity is therefore $\frac{4}{18-2} = 0.25$

1.15 Diameter

Definition: The diameter is the length of the longest path. Although loops are not mentioned in this definition, they pose a problem as a model with a loop can have infinitely long paths. To prevent this from happening, a loop is only allowed to be executed once. As a result, there are no repetitions of the entire loop in a path. The definition of a path is the same as explained in the example of section 1.14.

Related work: [2, 9, 10, 14]

Related aspects: Activities, events and data objects, gateways

Function name R package: diameter(file_path)

Interpretation: The higher the diameter, the less understandable the model.

Range: $[0, +\infty]$

Example: The longest paths are taken from the path log. The longest paths in the example are <S2, G1, A, G2, D, G1, A, G2, G3, G4, A, C, E, G5, C, E1> and <S2, G1, A, G2, D, G1, A, G2, G3, G4, A, C, F, G5, C, E1>. The length of these paths is 16. Therefore, the diameter is equal to 16.

1.16 Depth

Definition: Depth is a metric originating from software complexity metrics. It relates to the nesting of split and join constructs, which are also known as structured blocks. The depth of a model is defined as the maximum depth of all paths. The depth of a path is calculated by assigning $+1$ to a split and -1 to a join. The cumulative sum of these values is taken and the maximum level of the cumulative sum throughout all paths is the depth of the model. The assumption is made that the joins coming before the first split in a path are ignored.

Related work: [3, 8, 9, 11]

Related aspects: Gateways

Function name R package: depth(file_path)

Interpretation: The higher the depth, the less understandable the model.

Range: $[0, +\infty]$

Example: A small modification is made in the example to illustrate how to calculate the depth. This example can be found in figure 2. Suppose that the depth of the path <S2, G1, A, G2, G3, G4, A, C, G6 , G, G7, G5, C, E1 > is calculated.The calculation of the depth of the path is illustrated in table 1. The first row represents the path. The second row contains the assigned values according to the definition. The third row contains the cumulative sum. This contains the cumulative sum of the assigned values. Lastly, the maximum of the cumulative sum is taken. Therefore, the depth of the path is equal to 2. The depth of the model is the maximum of the depth of all paths, which is in this case also equal to 2.

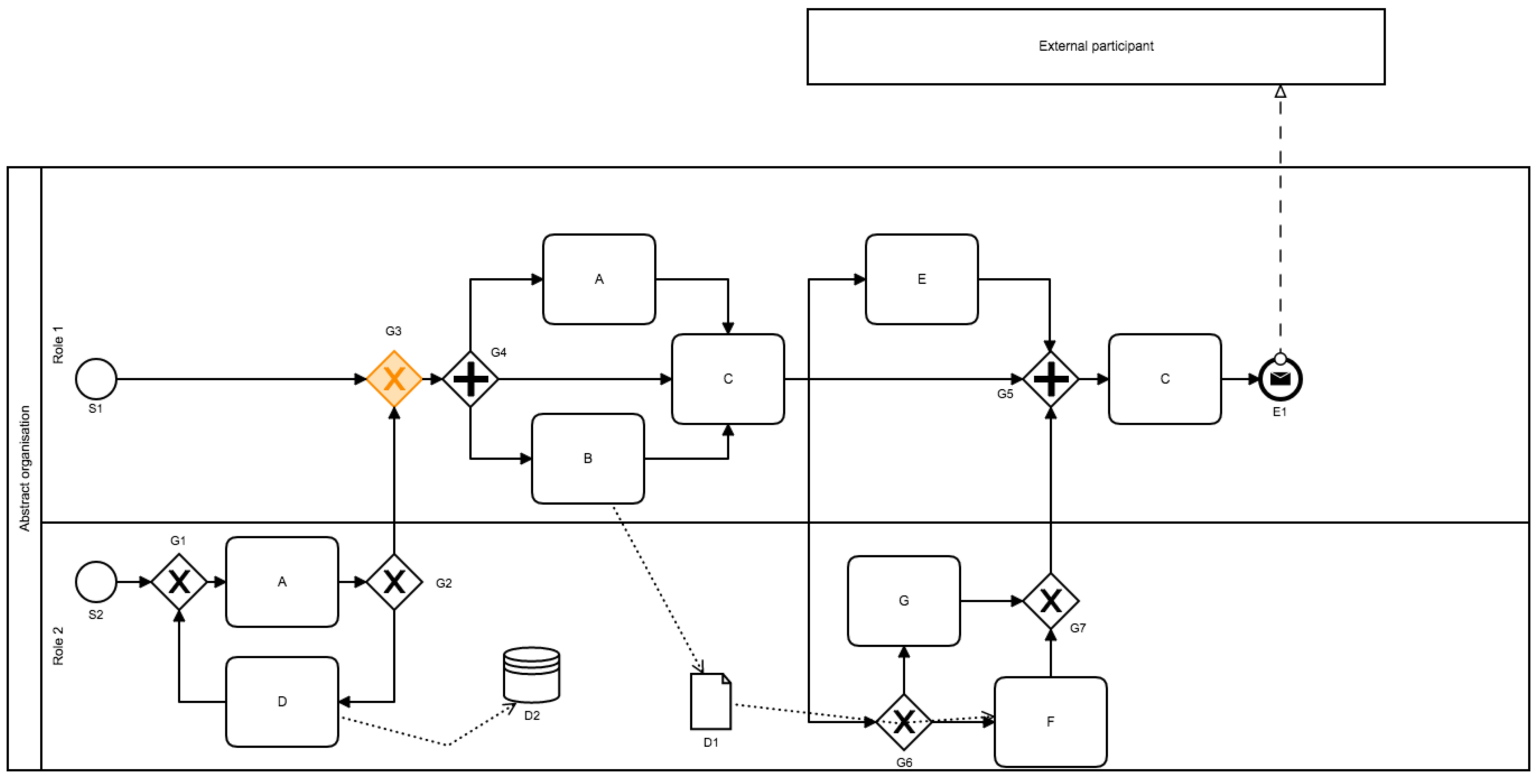

Figure 2: Modified example to calculate metrics

Path	S2	G1	A	G2	G3	G4	A	C	G6	G	G7	G5	C	E1
Assign	/	/	/	+1	-1	+1	0	-1 + 1	+1	0	-1	-1	0	0
Cumsum	0	0	0	+1	0	+1	+1	+1	+2	+2	+1	0	0	0
Max	0	0	0	+1	+1	+1	+1	+1	+2	+2	+2	+2	+2	+2

Table 1: Illustration calculation of depth of path

1.17 Token split

Definition: The token split gives an indication of the level of activities that are executed in parallel and measures therefore the concurrency. As the tokens are only split in inclusive and parallel gateways and activities with multiple outgoing sequence flows, the token split is calculated by summing the number of outgoing flows from these objects and subtracting one for each object.

Related work: [9, 10, 14]

Related aspects: Gateways, flows

Function name R package: concurrency(file_path)

Interpretation: The higher the token split, the less understandable the model.

Range: $[0, +\infty]$

Example: Figure 1 is again used as an example. G4 and C are the relevant objects for calculating the token split. Both objects have three outgoing sequence flows. The token split is thus 6 - 1 - 1 = 4.

1.18 Control flow complexity

Definition: The control flow complexity is a measure introduced by Cardoso [1]. It takes into account that an inclusive gateway is more difficult to understand than a parallel gateway or exclusive gateway by assigning a higher weight to this element. The control flow complexity is defined as follows:

$$\sum\nolimits_{AND\ gateways\ and\ split\ activities} 1 +$$

$$\sum\nolimits_{XOR\ gateways} |XOR\ gateways \bullet| +$$

$$\sum\nolimits_{OR\ gateways} 2^{|OR\ gateways \bullet|}$$

where $|object \bullet|$ refers to the outgoing sequence flows of object

Thus, the control flow complexity contains the number of parallel gateways and activities with multiple outgoing sequence flows, the sum of all outgoing sequence flows for all exclusive gateways and two to the power of the number of outgoing sequence flows summed for all inclusive gateways. As a result, exclusive gateways decrease the understandability more than parallel gateways, and inclusive gateways decrease the understandability even more than exclusive gateways do. Note that only split gateways are used in this definition, as the join gateways do not make the behaviour of the process model more complex.

Related work: [1, 3, 9, 14]

Related aspects: Gateways, flows

Function name R package: control_flow_complexity(file_path)

Interpretation: The higher the control flow complexity, the less understandable the process model.

Range: $[0, +\infty]$

Example: In our example, only three objects are relevant. These are the gateways G2 and G4 and the activity C. As G2 is an exclusive gateway with two outgoing sequence flows, the corresponding value equals to 2. G2 and C receive a value of 1. The control flow complexity is therefore 2 + 1 + 1 = 4.

1.19 Connector mismatch

Definition: Connector mismatch is defined as the sum of the absolute value of the difference between the number of split gateways and the number of join gateways for each gateway type. An activity with multiple outgoing sequence flows is considered a parallel split in this definition. In addition, an activity with multiple incoming sequence flows is considered an exclusive join.

Related work:[3, 9, 11, 14]

Related aspects: Gateways

Function name R package: connector_mismatch(file_path)

Interpretation: The bigger the connector mismatch, the less understandable the process model.

Range: $[0, +\infty]$

Example: G1, G3 and C are considered an exclusive join gateway. C is considered an exclusive join gateway, because there are several incoming sequence flows. G2 is an exclusive split gateway. G4 and C are considered as parallel split gateways. G5 is a parallel join gateway. The absolute value of the difference of exclusive gateways is 3 - 1 = 2. The absolute value of the difference of parallel gateways is 2 - 1 = 1. The connector mismatch is the sum and hence 2 + 1 = 3.

1.20 Connector heterogeneity

Definition: The connector heterogeneity gives the entropy over the gateway types and keeps into account the possibility that some gateways may not be present in the model. This metric is inspired by the information entropy. For calculating the connector heterogeneity, the relative frequency of the gateways for each gateway type is calculated using the following formula:

$$p_x = \frac{\#gateways\ of\ type\ X}{\#\ all\ gateways}$$

The connector heterogeneity is then defined as follows:

$$connector\ heterogeneity = -\sum\nolimits_{alltypes} p_x \times \log_{\#\ types}(p_x)$$

In this definition, Log is the mathematical operator logarithm. The base number is equal to the number of types of gateways. When there is exactly one type of gateway present in the model or if there are no gateways, the connector heterogeneity is set to 0. In addition, the activities with multiple outgoing sequence flows are considered a parallel split gateway. The activities with multiple incoming sequence flows are recognized as exclusive join gateways.

Related work: [2, 9, 10, 14]

Related aspects: Gateways, flows

Function name R package: connector_heterogeneity(file_path)

Interpretation: The higher the connector heterogeneity, the less understandable the process model.

Range: $[0, +1]$

Example: In the example, there are three exclusive gateways, two parallel gateways, one activity with multiple incoming sequence flows and one activity with multiple outgoing sequence flows. Therefore, $p_{XOR} = \frac{4}{7}$ and $p_{AND} = \frac{3}{7}$. The number of types equals two, which is the base number of the logarithm. The connector heterogeneity is equal to $-\frac{4}{7} \cdot log_2(\frac{4}{7}) - \frac{3}{7} \cdot log_2(\frac{3}{7}) =$ 0.99

1.21 Separability

Definition: Separability measures how easy it is to split the model into several independent parts. It is defined as

$$\frac{\#\ cut\ vertices}{process\ model\ size - \#\ data\ objects - 2}$$

A cut vertex is a sequence flow, which if removed, separates the model in two different models which have no connection any more. The size is the process model size as defined in section 1.1.

Related work: [9, 10, 14]

Related aspects: Activities, events and data objects, gateways, flows, partitionability

Function name R package: separability(file_path)

Interpretation: The higher the separability, the more understandable the model.

Range: $[0, \frac{process\ model\ size}{process\ model\ size - 2}]$

Example: To find the cut vertices in the path log, the intersection of the elements between all paths is taken. These paths are the paths resulting from the path log described in section 1.14. If we take for example the path <S1, G3, join, G4, split, A, C, join, split, E, G5, join, C, E1> and the path <S2, G1, join, A, G2, split, D, G1, join, A, G2, split, G3, join, G4, split, A, C, join, split, E, G5, join, C, E1>, the elements they have in common are G3, G4, A, C, E, G5, C, E1. Split and join are left out, because they are not relevant. Instead of taking the intersection between two paths, the intersection between all paths is taken. The result is a list of all elements the paths have in common. In the example of figure 1, the intersection between all paths is G3, G4, C, G5, C and E1.

Then, the elements which are both a split and a join are filtered, because these elements cannot be a cut vertex. This follows from the fact that the model is not separated into two different models when one sequence flow is removed. The removed element in this case is C, because this activity has multiple incoming and outgoing sequence flows. Afterwards, the sequence flows between the remaining elements are identified. These are the flows between G3 and G4, between G5 and C and between C and E1. They are the cut vertices and total to three. The process model size was calculated in section 1.1 and equals 18. The sequentiality is then $\frac{3}{18-2-2} = 0.21$.

1.22 Extended cyclomatic metric of McCabe

Definition: The extended cyclomatic metric is defined as the sum of the flows minus the sum of the nodes plus the the sum of the strongly connected components in the reachability graph of a process model. The flows represent activities in the reachability graph. The nodes are the exclusive gateways at level zero which are not directly proceeded by an exclusive split. The strongly connected components are defined on the basis of exclusive split and join gateways. The number of connected components is equal to the number of exclusive gateways at depth level zero. This means that only the exclusive gateways which are not proceeded by an exclusive split gateway unless that exclusive split gateway is first followed by an exclusive join are considered. Also the tasks at depth level zero, which means that they are not proceeded by an exclusive split unless this split is followed by a join before the task are considered. Only the tasks directly before exclusive splits are discarded. The number of strongly connected components is in that case the sum of the number of exclusive split gateways at level zero and the sum of the number of tasks at level zero without the tasks directly proceeded by an exclusive split. In this definition an exclusive join which is the result of a loop is considered an exclusive split gateway and an exclusive split which is the result of a loop is considered an exclusive join gateway. This definition can be simplified to number of tasks - number of exclusive gateways and tasks which act as an exclusive join gateway + number of exclusive split gateways at level zero.

Related work: [7]

Related aspects: Activities, events and data objects, gateways

Function name R package: cyclomatic_metric(file_path)

Interpretation: The higher the extended cyclomatic metric of McCabe, the less understandable the model.

Range: $[0, +\infty]$

Example: The number of tasks is equal to 8 eight in the example process model. The number of exclusive gateways or activities which act as an exclusive gateway is four. These are G1, G2, G3 and the first activity C. The number of exclusive split gateways at level zero is 1 (G2). The extended cyclomatic metric of McCabe is therefore $8 - 4 + 1 = 5$.

1.23 Structuredness

Definition: The structuredness measures how the process model can be decomposed into structured blocks. A structured block is a block which starts with a split gateway and has a matching join gateway. In order to measure the structuredness, a reduced model is used. This reduced model is based on several reduction rules. To calculate the structuredness, the following formula is used:

$$1 - \frac{size_{reduced}}{size_{full} - \#\, data\ objects}$$

The following rules are used to retrieve the reduced model:

1. Delete all trivial constructs. A trivial construct is a node which has at most one incoming and one outgoing sequence flow. A node can be a gateway (even though this is unlikely), an activity or an event. An example of a trivial construct can be found in figure 3.

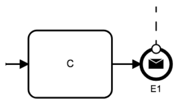

Figure 3: Trivial construct

2. Delete all split and join gateways or tasks which are the consequence of having multiple start or end events. An example is provided in figure 4.

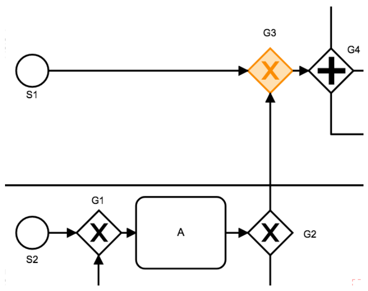

Figure 4: Join due to multiple start events

3. Delete all matching join and split gateways. This is done by assigning the according gateway type to each split and join. A distinction is made between gateways caused by a loop and other gateways. For gateways caused by a loop, only matching splits and joins of the type exclusive gateway are deleted. Matching splits and joins of gateways not caused by a loop are deleted as well.

Related work: [9, 10, 14]

Related aspects: Activities, events and data objects, gateways

Function name R package: structuredness(file_path)

Interpretation: The higher the structuredness, the more understandable the model.

Range: $[0, 1]$

Example: First the path log is created in a similar way as explained in section 1.14. However, the split and join elements are replaced with more detailed categories. The following categories are defined: XOR-loop-split, XOR-loop-join, other-loop-split, other-loop-join, start-join, end-split, AND-split, AND-join, OR-split, OR-join, XOR-join and XOR-split. In this example, G2 is a XOR-loop-split and G1 is a XOR-loop-join. G3 is a start-join. G4 is an AND-split and C receives the same category. However, C receives the category XOR-join as well. Lastly, the gateway G5 is an AND-join.

When all splits and joins are assigned to a category, the reduction rules are performed on each path. We use the path <S2, G1, join, A, G2, split, D, G1, join, A, G2, split, G3, join, G4, split, A, C, join, split, E, G5, join, C, E1> as an example. If the splits and joins are replaced with the categories, we get < S2, G1, XOR-loop-join, A, G2, XOR-loop-split, D, G1, XOR-loop-join, A, G2, XOR-loop-split, G3, start-join, G4, AND-split, A, C, XOR-join, AND-split, E, G5, AND-join, C, E1>. First all trivial constructs are removed from the path. Trivial constructs are elements in the path which are not followed by some type of join or split. S2, A, E, C and E1 are in the example path trivial constructs and are removed. The result is: < G1, XOR-loop-join, G2, XOR-loop-split, G1, XOR-loop-join, G2, XOR-loop-split, G3, start-join, G4, AND-split, C, XOR-join, AND-split, G5, AND-join >. Then, the start-join and end-split gateways are removed. In this case G3 is a start-join gateway. Now, the matching and split join gateways need to be removed. To do this, the first non-loop join or the first loop-split is identified. This is G2, a XOR-loop-split. The element before G2 is G1, which is a XOR-loop-join. As these gateways are matching, these can be safely removed. This reasoning is applied again and the result is: < G4, AND-split, C, XOR-join, AND-split, G5, AND-join >. When we continue, the first join is activity C. This activity behaves as a XOR-join, while the proceeding gateway is an AND-split. These are added to a list of objects from the reduced model and are removed in the path. There is nothing wrong with the match of C and G5 and they can thus be safely removed. The list of objects from the reduced model contains two elements: G4 and C. The same reasoning is applied to all paths. The unique elements in the list of objects of the reduced model is the size of the reduced model. In this case, only the two elements G4 and C are part of the reduced model. The size is thus 2. The process model size is 18. The structuredness is $1 - \frac{2}{18-2} = 0.875$

1.24 Cross-connectivity

Definition: The cross-connectivity metric tries to measure the strength of the sequence flows between all nodes, which are gateways and activities. It does this by assigning a weight to each sequence flow. Even though the cross connectivity is inspired by the control flow complexity, the control flow complexity takes no entire paths into consideration, while the cross connectivity does. To calculate the cross connectivity, all nodes in the process model receive a weight (W) with d the number of incoming and outgoing sequence flows of a node n:

$$W(n) = \begin{cases} 1 & \text{if n is an AND-gateway or task} \\ \dfrac{1}{d} & \text{if n is an XOR-gateway} \\ \dfrac{1}{2^d - 1} + \dfrac{2^d - 2}{2^d - 1} \cdot \dfrac{1}{d} & \text{if n is an OR-gateway} \end{cases}$$

The weight of a sequence flow is:

$$W(sf) = w(source(sf)) \cdot w(target(sf))$$

with w(source(sf)) the weight of the node which is the source of the sequence flow and w(target(sf)) the weight of the node which is the target of the sequence flow. The value of a path is defined as follows:

$$\prod_{i\ in\ path} W(sf_i)$$

In other words, the value of a path is a product of the weights of the sequence flows. The value of a connection is the maximum value of all paths between node n_1 and n_2. The cross connectivity metric is defined as follows:

$$\frac{\sum V(n_1, n_2)}{(process\ model\ size - \#\ data\ objects) \times (process\ model\ size - \#\ data\ objects - 1)}$$

with V the value of a connection between n_1 and n_2

Related work: [15]

Related aspects: Activities, events and data objects, gateways, flows

Function name R package: cross_connectivity(file_path)

Interpretation: The higher the cross connectivity, the more understandable the process model

Range: $[0, 1]$

Example: We will show how to calculate the value of a path between S1 and E1. There are 7 relevant nodes: G3, G4, A, C, E, G5 and C in the path <S1, G3, G4, A, C, E, G5, C, E1 >. The path with the elements G3, G4, A, C, F, G5 and C will yield the same value. The elements G4, A, E, G5 and the second C have the weight 1, because they are activities or AND-gateways. The first C is considered a XOR-join, together with G3. The corresponding weights are $\frac{1}{3}$ and $\frac{1}{2}$. Therefore, the value of the path is $1 \times \frac{1}{2} \times 1 \times \frac{1}{3} \times 1 \times 1 \times 1 = \frac{1}{6}$, which is also the value of the connection between S1 and E1. There are no different paths between these two nodes with a higher value. The sum of the values of the connections between all nodes is taken and totals 23.24. The cross connectivity is then $\frac{23.24}{(18-2)*(17-2)} = 0.096$.

To calculate the value of all paths, the path log is used. An iteration over all elements is performed in order that the values of all sub paths can be calculated. Suppose that a path contains 5 nodes. Then, the value of the path $< n_1, n_2 >$, $< n_1, n_2, n_3 >$, $< n_1, n_2, n_3, n_4 >$ and $< n_1, n_2, n_3, n_4, n_5 >$ is calculated. Each value is saved, together with the start and end node of the sub path. The maximal values for each start and end node are taken in order to get the value of the connections. Then the sum can be calculated as explained in the previous paragraph.

References

[1] Jorge Cardoso. Control-flow complexity measurement of processes and Weyuker's properties. In *6th International Enformatika Conference*, volume 8, pages 213–218, 2005.

[2] María Fernández-Ropero, Ricardo Pérez-Castillo, Ismael Caballero, and Mario Piattini. Quality-driven business process refactoring. In *International Conference on Business Information Systems (ICBIS 2012)*, pages 960–966, 2012.

[3] Kathrin Figl. Comprehension of Procedural Visual Business Process Models: A Literature Review. *Business & Information Systems Engineering*, 59(1):41–67, February 2017. ISSN 2363-7005, 1867-0202. doi: 10.1007/s12599-016-0460-2. URL `http://link.springer.com/10.1007/s12599-016-0460-2`.

[4] V. Gruhn and R. Laue. Reducing the cognitive complexity of business process models. pages 339–345, June 2009. doi: 10.1109/COGINF.2009.5250717.

[5] Thomas Gschwind, Jana Koehler, and Janette Wong. Applying patterns during business process modeling. *Business process management*, pages 4–19, 2008.

[6] Marcello La Rosa, Petia Wohed, Jan Mendling, Arthur HM Ter Hofstede, Hajo A. Reijers, and Wil MP van der Aalst. Managing process model complexity via abstract syntax modifications. *IEEE Transactions on Industrial Informatics*, 7(4):614–629, 2011.

[7] Kristian Bisgaard Lassen and Wil MP van der Aalst. Complexity metrics for Workflow nets. *Information and Software Technology*, 51(3): 610–626, 2009.

[8] Ralf Laue and Volker Gruhn. Complexity Metrics for business Process Models. In *Business Information Systems*, Klagenfurt, Austria, January 2006.

[9] Jan Mendling. *Detection and prediction of errors in EPC business process models*. PhD thesis, Wirtschaftsuniversität Wien Vienna, 2007. URL `http://www.mendling.com/publications/Mendling%20Doctoral%20thesis%20extract.pdf`.

[10] Jan Mendling and Mark Strembeck. Influence Factors of Understanding Business Process Models. In *Lecture Notes in Business Information Processing*, volume 7, pages 142–153, May 2008. doi: 10.1007/978-3-540-79396-0_13.

[11] Geoffrey Muketha. Complexity Metrics for Measuring the Understandability and Maintainability of Business Process Models using Goal-Question-Metric (GQM). *International Journal of Computer Science and Network Security*, 8(5):219–225, 2008.

[12] Razvan Petrusel, Jan Mendling, and Hajo A. Reijers. How visual cognition influences process model comprehension. *Decision Support Systems*, 96(Supplement C):1–16, April 2017. ISSN 0167-9236. doi: 10.1016/j.dss.2017.01.005. URL `http://www.sciencedirect.com/science/article/pii/S016792361730012X`.

[13] Gregor Polančič and Blaž Cegnar. Complexity metrics for process models – A systematic literature review. *Computer Standards & Interfaces*, 51(Supplement C):104–117, March 2017. ISSN 0920-5489. doi: 10.1016/j.csi.2016.12.003. URL `http://www.sciencedirect.com/science/article/pii/S0920548916302276`.

[14] Hajo A. Reijers and Jan Mendling. A study into the factors that influence the understandability of business process models. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(3):449–462, 2011.

[15] Irene Vanderfeesten, Hajo A. Reijers, Jan Mendling, Wil M. P. Aalst, and Jorge Cardoso. On a Quest for Good Process Models: The Cross-

Connectivity Metric. Montpellier, France, June 2008. doi: 10.1007/978-3-540-69534-9_36.