Adopting the Cognitive Complexity Measure for Business Process Models

Volker Gruhn, Ralf Laue
University of Leipzig, Germany, Chair of Applied Telematics / e-Business
{gruhn, laue}@ebus.informatik.uni-leipzig.de

Abstract

Business process models, often modelled using graphical languages like UML, serve as a base for communication between the stakeholders in the software development process. To fulfill this purpose, they should be easy to understand and easy to maintain.

For this reason, it is useful to have measures that can give us some information about understandability, analyzability and maintainability of a business process model.

Shao and Wang have proposed a cognitive complexity measure[19]. It can be used to estimate the comprehension effort for understanding software. This paper discusses how these research results can be extended in order to analyze the cognitive complexity of graphical business process models.

Keywords: cognitive complexity, complexity metrics, business process models

1 Introduction

One of the main purposes for developing business process models is to support the communication between the stakeholders in the software development process (domain experts, business process analysts, software developers to name just a few). To fulfill this purpose, the models should be easy to understand and easy to maintain. If we want to create models that are easy to understand, at first we have to define what "easy to understand" means: We are interested in complexity metrics, i.e. measurements that can tell us whether a model is easy or difficult to comprehend. In the latter case, we may conclude from the metrics that the model should be re-engineered, for example by decomposing it into simpler modules.

A significant amount of research has been done on the complexity of software programs, and software complexity metrics have been used successfully for purposes like predicting the error rate, estimating maintenance costs or identifying pieces of software that should be re-engineered. However, to our best knowledge, there is no much published work about complexity of business process/workflow models. This related work is discussed in Sect. 3. In Sect. 4, we shortly introduce the cognitive weight for software defined in [19], and in Sect. 5, we show how these ideas can be adopted for being used with business process models. The limitations of this approach will be discussed in Sect. 6, and Sect. 7 concludes and shows directions for further research.

2 Business Process Models

A business process model (BPM) can describe various aspects of a business process like the control flow, the exchange of documents, used resources and responsibilities.

In this paper, our focus is the control flow which can be visualized using existing BPM modeling languages. The control flow describes the order in which activities (tasks) are processed. The most simple case of a BPM is a model where activities have to be processed in a sequence. However, most BPMs contain split nodes (to start the parallel execution of concurrent paths or to select one or more paths to process) and join nodes (to synchronize concurrent paths or to join mutually exclusive paths) as well. Split and join nodes can be used to model parallel or alternative executions of control flow paths. A formal definition of BPMs can be found for example in [1].

3 Related Work

To our knowledge, there is (apart from our own paper [7]) no much published work about complexity analysis of workflows / BPMs.

The authors of [14] suggest several measures and test their significance for the purpose of predicting errors in a BPM. Although the most metrics discussed in [14] are simple counting metrics ("number of a certain kind of elements in a model") which do not give much information about understandability of the model as a whole, the results from this
work are valuable to draw conclusions about the difficulty to comprehend different modeling elements.

Cardoso[6] has suggested a complexity measure for BPMs which is a generalization of McCabe’s cyclomatic complexity for software[12]. The control-flow complexity of processes (CFC) as defined by Cardoso is the number of mental states that have to be considered when a designer develops a process. The CFC metric counts the number of decisions in the flow of control. These decisions take place at the split nodes which model alternative or parallel routing in the control flow. There are three different types of split nodes which add to the number of possible decisions as follows:

– AND-split (All paths outgoing from an AND-split must be processed): The designer needs only to consider one state (processing of all outgoing paths) as the result of the execution of an AND-split. For this reason, every AND-split in a model adds 1 to the CFC metric of this model.

– XOR-split with n outgoing paths (Exactly one from n possible paths must be taken): The designer has to consider n possible states that may arise from the execution of the XOR-split. For this reason, every XOR-split with n outgoing paths adds n to the CFC metric of this model.

– OR-split with n outgoing paths (One or several of the n possible paths must be taken): There are \(2^n - 1\) possibilities to process at least one and at most n of the outgoing paths, i.e. every OR-split with n outgoing paths adds \(2^n - 1\) to the CFC metric.

If we define complexity as "difficulty to test" (i.e. number of test cases needed to achieve full path coverage), the CFC metric does a perfect job. However, we argue that this metric is less useful if we define complexity as "difficulty to understand a model": For example, the number of control flow paths between an OR-split and an OR-join has not much influence on the effort that is necessary to comprehend this control structure. Regardless of whether there are 2, 4 or 10 control flow paths between the split and the join, the person who reads the model will always understand everything between the OR-split and the OR-join as a whole, i.e. everythin between the OR-split and the OR-join is considered as a unit.

To overcome these shortcomings, we suggest to adapt the cognitive weight measure defined by Shao and Wang which is intended to measure the cognitive and psychological complexity of software as a human intelligence artifact[19].

<table>
<thead>
<tr>
<th>control structures</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence (an arbitrary number of statements in a sequence without branching)</td>
<td>1</td>
</tr>
<tr>
<td>call of a user-defined function</td>
<td>2</td>
</tr>
<tr>
<td>branching with if-then or if-then-else</td>
<td>2</td>
</tr>
<tr>
<td>branching with case (with an arbitrary number of selectable cases)</td>
<td>3</td>
</tr>
<tr>
<td>Iteration (for-do, repeat-until, while-do)</td>
<td>3</td>
</tr>
<tr>
<td>recursive function call</td>
<td>3</td>
</tr>
<tr>
<td>execution of control flows in parallel</td>
<td>4</td>
</tr>
<tr>
<td>Interrupt</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1. Cognitive Complexity Metrics

Shao and Wang [19] defined the cognitive weight as a metric to measure the effort required for comprehending a piece of software. Based on empirical studies, they defined cognitive weights for basic control structures. Table 1 shows the basic control structures and their cognitive weights W which are a measure for the difficulty to understand a software control structure.

The cognitive weight of a software component without nested control structures is defined as the sum of the cognitive weights of its control structures according to Table 1. It seems to be a promising approach to use the ideas from [19] to define cognitive weights for BPMs. If we want to do so, we have to consider some important differences between graphical BPMs and software programs.

At first, we mention that for BPMs, we can ignore a fact that is very important for the cognitive complexity of software. The complexity of software depends on three factors: internal processing, input and output[20]. The cognitive size of a software component is a function of the cognitive weight of its control structures and its inputs and outputs (i.e. arguments in a function call and return values delivered by a function). However, often BPMs show the flow of control only and abstract away from inputs and outputs (called document flow in the BPM terminology). For this reason, we will not have to take into account input and output in our definitions.

Furthermore, Table 1 should be tailored to the demands of BPMs: While recursion does not play any role in business process modeling, it is necessary to consider other concepts like cancellation or the multi-choice-pattern[3]. This will be discussed in section 5.

Once we have defined the cognitive weights of the elements of a BPM (i.e. its control structures as they will be defined in section 5), we define the cognitive weight of a BPM as follows:

\[1\] Note that this definition is slightly different from the definition of the
Definition 1 (Cognitive weight of a BPM): The cognitive weight of a BPM is the sum of the cognitive weights of its elements.

5 Cognitive Weight of BPM Modeling Elements

In this section, we will define the cognitive weight of the "building blocks" of a BPM that is modeled in a graphical language. For such models, several BPM languages (for example BPMN[5] or UML Activity Diagrams[15]) have been proposed which differ in their expressiveness. We have chosen to study the modeling elements of the BPM language YAWL[2] because of its expressive power. YAWL has been developed based on the results from workflow patterns research[3]. It enables the modeler to build BPMs that include all complex workflow patterns using easy-to-understand symbols.

Table 2 shows all YAWL symbols and relates them to corresponding software control structures. For these software control structures, cognitive weights have been defined in [19]. We assume that these cognitive weights are also a good measure for the difficulty to comprehend the corresponding BPM element. This assumption is based on the insight that the behavior of a software system as well as the behavior modeled by a business process can be described in the same formal way.[21]

For some BPM control structures, the mapping to a corresponding software control structure is straightforward (see Table 2). For example, a sequence in a BPM (a number of consecutive steps in a workflow) and a sequence in a piece of software (a number of consecutive instruction statements) require the same effort to comprehend, and a composite task in a YAWL-model is nothing else than a call of a user-defined function (modelled on another level of abstraction). However, other control structures (the ones with a reference in the last column in Tab. 2), require some distraction). However, other control structures (the ones with a reference in the last column in Tab. 2), require some distraction). However, other control structures (the ones with a reference in the last column in Tab. 2), require some distraction). However, other control structures (the ones with a reference in the last column in Tab. 2), require some distraction). However, other control structures (the ones with a reference in the last column in Tab. 2), require some distraction). However, other control structures (the ones with a reference in the last column in Tab. 2), require some distraction). However, other control structures (the ones with a reference in the last column in Tab. 2), require some distraction).

5.1 OR-Splits/-Joins

In BPMs, a control block starting with an OR-split with \( n \geq 2 \) outgoing branches and ending with an OR-join is used to model the parallel execution of at least one and at most \( n \) branches in parallel. There is no corresponding software control structure. However, if we look at what exactly is done at an OR-split, we can see that this is nothing else than

The first step is equivalent to the software control structure "Case" (which has the cognitive weight 3), the second step is equivalent to the software control structure "Parallel" with the cognitive weight of 4. For this reason, we define the cognitive weight of an OR-branching construct in a BPM as \( 3+4=7 \). Note that this cognitive weight does not depend on the number of branches between the OR-split and the OR-join.

From Table 2, we see that OR-constructs with a cognitive weight of 7 have the largest complexity among all BPM control structure elements. This may look surprising at first glance, but is backed by the research results by Sarshar and Loos who showed in laboratory experiments that OR-constructs in BPMs were significantly less comprehended than AND- and XOR-control flow blocks[18]. Also, in [14] it has been shown that OR-join connectors in BPMs have the highest impact on the odds of an error in a BPM modelled as Event-Driven Process Chain (compared to XOR- and AND-connectors).

5.2 Multiple Instantiation

Multiple instances are used in BPMs if an activity in a BPM can have more than one running instance at the same time. For example, in a model of the review process of a scientific paper, an activity "write review" is executed multiple times at the same time (by different reviewers). Similar to the OR-split discussed in 5.1, we mention that processing a multiple instance task involves two steps:

1. choosing the number of activities to be activated in parallel
2. processing these activities in parallel

The first step is the simple choice of an integer number, i.e. only one simple decision. For this reason, we assume that it has the same cognitive complexity as a simple branching decision with \( \text{if-then} \) which has a cognitive weight of 2. Even if the choice of an integer may have more than two possible outcomes (as this is the case with \( \text{if-then} \)), we argue that the complexity is lower than that of the case control structure, because a case-statement is followed by \( \geq 3 \) different executions which have to be comprehended. In contrast to this, a multiple instantiation means that only one kind of activity has to be performed several times in parallel.

For the mentioned reasons, we define the cognitive weight of activities that allow multiple instantiation as the
<table>
<thead>
<tr>
<th>Workflow Pattern[3]</th>
<th>BPM control structure</th>
<th>corresponding software control structure</th>
<th>YAWL-Symbol</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>consecutive steps in a workflow</td>
<td>sequence</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Exclusive Choice</td>
<td>XOR-split (exactly one of two branches is chosen) with corresponding XOR-join</td>
<td>branching with if-then</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>XOR-split (exactly one of ( \geq 3 ) branches is chosen) with corresponding XOR-join</td>
<td>branching with case (with an arbitrary number of selectable cases)</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Parallel Split and Synchronization</td>
<td>An AND-split activates all outgoing links in parallel, a corresponding AND-join synchronizes the flows of control</td>
<td>execution of control flows in parallel</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Multiple Choice and Synchronizing Merge</td>
<td>OR-split (a number of branches is chosen from 2 or more possible branches) with corresponding OR-join</td>
<td>branching with case, followed by parallel execution</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>(none)</td>
<td>Composite task (subtask, can be used for decomposing a BPM into modules)</td>
<td>call of a user-defined function</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Multiple Instances Patterns</td>
<td>Multiple Instance Activity (allows multiple instances of an activity to run concurrently)</td>
<td>branching, followed by parallel execution</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Cancel Activity</td>
<td>Cancellation (by activating an activity one deactivates another one)</td>
<td>see discussion in 5.3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Cancel Case</td>
<td>Cancellation (by activating an activity one deactivates all elements within another part of the model)</td>
<td>comparable to a function call</td>
<td></td>
<td>2 or 3</td>
</tr>
</tbody>
</table>

There are two cancellation patterns, "Cancel Activity" and "Cancel Case"[3]. In the "Cancel Activity" pattern, a single activity is deactivated. An example is shown in Fig. 1: By activating the activity "Withdraw Request", another activity "Process Request" is canceled. Afterwards, the business process continues with another activity "Write Log". The cancellation can be regarded as a "sidestep" before the normal processing proceeds, at runtime the cancellation becomes just one more step in a sequence (see Fig. 2). We define that the cognitive complexity of such a "sidestep" just adds 1 to the complexity of the model.

The "Cancel Case" pattern is more involved: By activating an activity, all instances of a number of activities are

---

2We do not discuss the differences in the patterns involving multiple instances here, for example whether it is possible to create new instances at run-time[3], because we assume that these differences do not have a major influence on the cognitive complexity.

sum of the cognitive weights of an IF-branching (2) and a parallel execution (4)².

5.3 Cancellation

Cancellation in a BPM means that by activating one task all elements within some part of the model are deactivated (i.e., activities are deleted from the list of activities to process or stopped if they are already running).
deactivated. For example, activation of an activity called "Cancel Order" may deactivate a number of corresponding activities like "Pack Order", "Ship Order" and "Send Invoice", before the business process continues, for example with an activity "Log Canceled Order". This is more than just a single "sidestep" in the "Cancel Activity" pattern. Instead, the cancellation of a case can be compared with a function call: At some step in the processing of the business process, the control is passed onto the "function" that does the cancellation, afterwards the proceeding steps of the business process are executed. For this reason, the cognitive weight of a "Cancel Case" element is defined to be equal to the cognitive weight of the "Function Call" control structure, which is 2.

However, there is one more fact to consider: The elements to cancel may contain activities with multiple instances or "nested elements" (i.e. nested UML activity diagrams, sub-processes in BPMN or composite tasks in YAWL). Because all instances of multiple instance activities must be deactivated and all "child tasks" of composite tasks must be deactivated at the same time, the reader of the model has to understand one more concept. For this reason, we add 1 to the cognitive weight of a "Function Call" structure if the elements to cancel (depicted by a dotted area in YAWL) contain a multi-instance task or a composite task.

6 Limitations of the approach

In this section, we discuss some aspects of complexity of BPMs that cannot be measured by the cognitive weight metric.

6.1 (Un)structuredness

The equivalence between XOR-splits in a BPM and the software control structure "branching with IF-THEN-ELSE" seems to be straightforward. However, at this point we have to remember that the cognitive weights for software control structures have been defined for being used with structured programming languages: A CASE- or IF-THEN-statement means that one of several possible instructions (which may include the "do nothing" instruction) will be selected and executed. Afterwards, the statement that follows the CASE-statement will be processed. In structured programming languages, there is no such thing as "IF expression THEN GOTO somewhere else". However, BPMs do not have to be modelled in such a structured way, they may contain arbitrary jumps out of and into control blocks.

It is well-known that such GOTO-statements make the code more difficult to read, and the same holds for graphical BPMs with GOTO-like jumps. The difficulties that arise from such jumps are not addressed by cognitive weight metric. For this reason, it is important to point out that the cognitive weight for BPMs as defined in this paper can be used only for such BPMs that are well-structured. Recent publications show that in most cases, unstructured models can be replaced by structured ones, and there are already BPM languages like BPEL4WS or the workflow management system ADEPT with semantic restrictions which force the modeler to build well-structured models.

6.2 Layout Complexity

The layout of a graphical model can contribute considerably to the difficulty to understand the model. However, such aspects cannot be measured with the metric discussed in this paper. Important considerations to good layout include for example:

- choosing size and color of the graphical elements in the model with care
- modeling time-dependency horizontally from left to right (at least for people who speak languages that read left-to-right) or vertically from top to bottom
- aligning the edges of the graphical elements
- avoiding intersecting arrows

While such points are important for drawing easy-to-read graphical BPMs. However, they are beyond the scope of this paper.

6.3 Textual Complexity

When defining the cognitive complexity as a measure of understandability of BPMs, our focus was the cognitive complexity defined by [1].
complexity of the control structures a BPM is built from. Implicitly, we assumed that every single activity is equally difficult to comprehend. However, as descriptions of activities in BPMs are usually given in natural language, this does not have to be the case. Instead, text comprehension plays an important role in the process of reading and understanding a BPM. For the purposes of reading and understanding text, the density of unfamiliar terms is a good indicator of the comprehension error. Klemola and Rilling have shown that these effects can also be used for measuring the complexity of software.

Such problems are not addressed by the cognitive complexity discussed in this paper. Instead, we regard it as a prerequisite for modeling that the people who work with the model agree about using an ontology, i.e. a certain vocabulary plus a set of assumptions regarding the intended meaning of the vocabulary words. It is good practice that a glossary, defining this ontology, is an integral part of a BPM.

7 Conclusion and Directions for Future Research

We expect that the cognitive weight as defined in this paper will be useful for measuring the complexity of BPMs. In our future research, we want to validate this expectation and compare the cognitive weight metric with other complexity metrics for BPMs. It may happen that the cognitive weights for the BPM building blocks have to be adjusted as a result of this research. We note that that the same validation still has to be done for the cognitive complexity measure for software as well, at least we do not know any related work on this topic. For example, it would be interesting to compare the weights defined in Tab. 1 with the context complexity weights suggested by McQuaid.

Due to the number of factors that contribute to the complexity of a BPM, we cannot expect that a single metric measures all aspects of a model’s complexity. This situation is well-known from the measuring of software complexity. A common solution is to associate different metrics within a metrics suite. Each individual metric in the suite measures one aspect of the complexity, and together they give a more accurate overview. Metrics that can be used for BPMs complementary to the cognitive metric are discussed in [7].

References