Metrics for the Case Management Modeling and Notation (CMMN) Specification

Mike A. Marin
University of South Africa
IBM Analytics Group
25131 Mackenzie Street
Laguna Hills, CA 92653, USA
mmarin@acm.org

Hugo Lotriet
College of Science
Engineering and Technology
University of South Africa
Florida Park, Johannesburg,
South Africa
lotrihh@unisa.ac.za

John A. Van Der Poll
Graduate School of Business
Leadership (SBL)
University of South Africa
Midrand, 1686, Gauteng,
South Africa
vdpolja@unisa.ac.za

ABSTRACT
The Case Management Modeling and Notation (CMMN) specification, published by the Object Management Group (OMG) in 2014, describes a declarative style for modeling business processes. The declarative nature of CMMN is intended to supplement the procedural style of the Business Process Modeling and Notation (BPMN). Although multiple metrics have been developed and verified for BPMN, the authors are not aware of any metrics developed for CMMN. Being a relative new process specification the understanding of complexity metrics for CMMN ought to be beneficial for practitioners and researchers by providing a way to compare case management models.

This study provides a formal description of CMMN and three metrics are defined, namely size, length, and complexity. The metrics are theoretically validated using the formal framework for software measurements defined by Briand et al. and the complexity metric is further validated using Weyuker’s properties for software complexity measures.

CCS Concepts
• General and reference → Metrics; • Applied computing → Business process modeling; • Software and its engineering → System modeling languages; • Computing methodologies → Model verification and validation;

Keywords
Case management, Case handling, CMMN, BPMN, Modeling complexity, Complexity metrics, Process modeling complexity

1. INTRODUCTION
This research defines three metrics for the Case Management Modeling Notation (CMMN) [25]. The proposed metrics are the size $CS$, the length $CL$, and complexity $CC$ of a CMMN model.

Most authors are in agreement that software metrics should be theoretically and empirically validated [22, 23, 30]. The metrics proposed in this study are theoretically validated using an axiomatic approach and future work will focus on the empirical validation. The Briand et al. framework [4] and Weyuker properties [36] are categorized as axiomatic or property-based approaches to validation [30], and they are commonly used for validating software metrics [25].

Business process improvement practitioners are increasingly interested in case management in many industries [14]. However, it is known that users find complex process models difficult to understand, and complex models are more likely to contain design errors [18]. Therefore, research on complexity metrics of process models is important, and several studies on the subject have been conducted [1, 5, 6, 11, 12, 31, 35, 18, 21, 26]. In those studies the researchers have used UML, BPMN, workflow nets, or proprietary BPM product models, all of which are based on directed graphs and are procedural in nature. CMMN defines a declarative style that is different to the BPM procedural style [19]. Therefore, the metrics defined for procedural models may not be applicable to CMMN. This study addresses that gap by proposing three new metrics for CMMN.

Section 2 provides brief background information about case management and the CMMN notation. Section 3 describes the methodology used, introduces a formal definition for CMMN models, and describes the three proposed metrics. Section 4 presents the theoretical validation for the proposed metrics using Briand et al. framework [4] and Weyuker properties [36]. Section 5 presents our findings and provides suggestions for future research. Finally, section 6 describes our conclusions.

2. BACKGROUND
Case management in the context of process technology was first introduced by Berkley and Eccles [2] in 1991 and Davenport and Nohria [8] in 1994. Case handling was introduced by Van Der Aalst and Berens in 2001 [33] and Reijers et al. [27] in 2003 to support the flexibility required by knowledge workers during a process and to help them better deal with exceptions that may occur during such process.

In 2009, the Object Management Group (OMG) issued a request for proposal (RFP) for the creation of a standard modeling notation for case management [24] to serve as a complement to its BPMN specification. The result was
CMMN [25], which was first published in 2014. The main difference between CMMN and BPMN is the shift from procedural to declarative models [19]. CMMN addresses case management as described in [33, 34, 27, 32] with a data-centric approach based on business artifacts [19]. Case management as defined by CMMN provides flexibility to knowledge workers with regards to what tasks or activities should be performed and when they should be performed [15].

Researchers are starting to look at CMMN and its applicability to case management requirements. Schöning et al. [28] in 2013 considered human centric processes starting with CMMN modeling skeletons that evolve over time. Marin et al. [20] in 2014 compared the CMMN method against other modeling notations. That study concluded that CMMN compares favorable to other process modeling notations like BPMN. Hauder et al. [13] in 2015 explored the applicability of CMMN for knowledge intensive processes exposed via a wiki environment for business users. The conclusion was that a wiki environment allows for non-technical users to structure knowledge intensive processes. Kurz et al. [17] in 2015 compared CMMN against adaptive case management and concluded that for the most part CMMN fulfills the adaptive case management requirements.

3. METHODOLOGY

This section starts by formalizing a CMMN model. We abstract and formalize the characteristics of a model that will allow for the definition of concrete metrics. The formalization will allow us to validate the metrics against formal frameworks like those of Briand et al. [4] and Weyuker [36].

Our focus is on complexity metrics for CMMN, and size and length are considered complexity metrics by Muketha et al. [23]. Therefore, we define three metrics, size $CS$, length $CL$, and complexity $CC$ of a CMMN model. The metrics are inspired by Briand et al. [4] definitions of size, length, and complexity. Briand et al. assume a system with procedural characteristics, and uses directed acyclic graphs to describe its framework. That imposed some constraints while defining the metrics, because CMMN is declarative instead of procedural.

The metrics are validated using the formal framework for software measurements defined by Briand et al. [4], and the complexity metric is further validated using the nine properties for complexity measures defined by Weyuker [36].

3.1 Case Management Modeling and Notation (CMMN)

DEFINITION 1. (Model): A CMMN [25] model $C$ is a collection of model elements $E$ with annotators $A$ and related by scope $U$ and event $V$ relationships. A model is defined as a tuple

$$C = (E, U, V, A)$$

Where

$E$ is a set of modeling elements. $E$ is strongly typed and each element on $E$ belongs to one of the following types: scope, data, plan, or optional. Table 1 shows the CMMN elements.

$U$ is a binary relationship in which two elements $x$ and $y$ in $E$ are related if and only if they are contained in the same scope. Note that $[x, y] \in U$ is an unordered pair.

### Table 1: CMMN elements

<table>
<thead>
<tr>
<th>type</th>
<th>Element ($E$)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>case</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stage</td>
<td></td>
</tr>
<tr>
<td></td>
<td>discretionary stage</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plan fragment</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>case file item</td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>discretionary task</td>
<td></td>
</tr>
<tr>
<td></td>
<td>event listener</td>
<td></td>
</tr>
<tr>
<td></td>
<td>milestone</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>connector (sentry)</td>
</tr>
</tbody>
</table>

$V$ is a binary relationship in which two elements $x$ and $y$ in $E$ are related if and only if an event from one $(x)$ triggers the other $(y)$. Note that $(x, y) \in V$ is an ordered pair.

$A$ is a set of annotators used to indicate characteristics of elements in $E$. Each annotator ‘a’ in $A$ is related to one and only one element $x$ in $E$. There are three types of annotators, namely decorators, sentries, and markers. Most elements $x$ in $E$ can be associated with a single marker, one of each decorator (collapsed or expanded), and multiple sentries. Table 2 shows the annotators.

DEFINITION 2. (Scope element set): A scope element is an element that can contain other elements. The set of scope elements $M$ is a subset of $E$ ($M \subseteq E$).

$$M = \{ z \mid \text{type-of}(z) \in \text{Scope} \}$$

Where, type-of($z$) is described by type in Table 1.

As shown in Table 1, the scope elements are the case, stage, discretionary stage, and plan fragment.

DEFINITION 3. (Case element set): A case element is a special scope element $z \in M$ that starts a case definition. A model $C$ can contain multiple case definitions, each one with its corresponding case element. The set of case elements $L$ is a subset of scope elements ($L \subseteq M$).

$$L = \{ z \mid \text{isa}(z) = \text{case} \}$$
Where, $\text{isa}(z)$ is described by name in Table 1.

Any nonempty model $\mathcal{C}$ must contain at least one case element. A case element can contain other elements, but it cannot be contained by any element.

Therefore, $\mathcal{C} \neq \emptyset \iff \mathcal{L} \neq \emptyset$.

**Definition 4. (Module):** A module of a model $\mathcal{C}$ is defined by a scope element $z \in \mathcal{M}$, as a tuple

$$\langle z \rangle = \langle \mathcal{E}, \mathcal{U}, \mathcal{V}, \mathcal{A} \rangle$$

(3.4)

Where

$\mathcal{E}_z$ is a subset of $\mathcal{E}$ defined as

$\mathcal{E}_z = \{ z \} \cup \{ x \mid x \text{ is reachable from } z \}$.

$\mathcal{U}_z$ is a subset of $\mathcal{U}$ defined as

$\mathcal{U}_z = \{ [\{ x, y \} \mid [x, y] \in \mathcal{U} \land x \in \mathcal{E}_z \land y \in \mathcal{E}_z \}$.

$\mathcal{V}_z$ is a subset of $\mathcal{V}$ defined as

$\mathcal{V}_z = \{ \{ x, y \} \mid \{ x, y \} \in \mathcal{V} \land x \in \mathcal{E}_z \land y \in \mathcal{E}_z \}$.

$\mathcal{A}_z$ is a subset of $\mathcal{A}$ defined as

$\mathcal{A}_z = \{ a \mid a \in \mathcal{A} \land (\exists y)(y \in \mathcal{E}_z \land y = \text{annotated-by}(a)) \}$

Where, annotated-by(a) describes the element in $\mathcal{E}$ that 'a' annotates.

Note that $z$ may contain other scope elements, in which case all the elements contained in these scope elements are also in $\langle z \rangle$. In other words, $x \in \mathcal{E}_z \land x \in \mathcal{M} \implies x \in \mathcal{E}_z$.

In addition, note that $\mathcal{E}$ is partitioned by $\mathcal{E}_z$ and $\mathcal{E}_y$,

$$\mathcal{E} = \mathcal{E}_z \cup \mathcal{E}_y \land \mathcal{E}_z \cap \mathcal{E}_y = \emptyset \iff \mathcal{L} = \{ x, y \}$$

(3.5)

By slight abuse of notation, we use $\mathcal{E}_P$ with upper case $P$ to indicate the set $\mathcal{E}$ of model $P$, and $\mathcal{E}_z$ with lower case $z$ to indicate the set $\mathcal{E}$ of module $\langle z \rangle$.

**Definition 5. (Scope relationship):** A scope relationship relates two elements if and only if they are in the same scope $z \in \mathcal{M}$. The scope is a binary relationship $\mathcal{U}$ on $\mathcal{E}$ ($\mathcal{U} \subseteq \mathcal{E} \times \mathcal{E}$), defined as follows

$$\mathcal{U} = \{ ([x, y] \mid x \neq y \land (\exists z)(z \in \mathcal{M} \land x \in \mathcal{E}_z \land y \in \mathcal{E}_z \\
\land x \neq z \land y \neq z) \\
\lor \neg(\exists w)(w \in \mathcal{M}_z \land (x \in \mathcal{E}_w \land y \in \mathcal{E}_w)) \}$$

(3.6)

Due to the declarative nature of CMMN, there is no strong sequence or control flow relationship between its elements, but scope plays an important role in a CMMN model.

**Definition 6. (Event relationship):** An event relationship relates two elements if and only if an event from one of them trigger an entry or exit criteria in the other element. The event is a binary relationship $\mathcal{V}$ on $\mathcal{E}$ ($\mathcal{V} \subseteq \mathcal{E} \times \mathcal{E}$), defined as follows

$$\mathcal{V} = \{ [x, y] \mid [x, y] \in \mathcal{U} \land x \xrightarrow{\text{event}} y \}$$

(3.7)

Events play an important role in CMMN, because entry and exit criteria for CMMN elements can be triggered by events. Events cannot cross scope, therefore the two modeling elements must already be related by scope.

<table>
<thead>
<tr>
<th>Type</th>
<th>Annotator (A)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>collapsed planing table</td>
</tr>
<tr>
<td>Decorator</td>
<td></td>
<td>expanded planing table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>auto complete</td>
</tr>
<tr>
<td>Sentry</td>
<td></td>
<td>entry criterion</td>
</tr>
<tr>
<td>Marker</td>
<td></td>
<td>non-blocking human</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exit criterion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>case</td>
</tr>
<tr>
<td></td>
<td></td>
<td>participant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>timer</td>
</tr>
</tbody>
</table>

Table 2: CMMN Annotators

**Figure 1: Model with one element (a case)**

### 3.2 Examples

Three examples of models are presented. The example models will be used to illustrate CMMN concepts and grammar. The models are grammatically and semantically correct, but they are not intended to represent any real world process. Annotators are not labeled in CMMN, but for illustration purposes, dotted lines were used to associate annotator labels to the annotator icons in the model.

**Example 1. Minimum case model.** Every model must contain at least one case element (see definition 3). Figure 1 shows the minimum valid CMMN model, with a single case element. The model has one model element $\mathcal{E} = \{ z \}$, one scope element $\mathcal{M} = \{ z \}$, and one case element $\mathcal{L} = \{ z \}$. The scope relationship is empty $\mathcal{U} = \emptyset$. The event relationship is empty $\mathcal{V} = \emptyset$. The annotator set is empty $\mathcal{A} = \emptyset$. This example contains a single module $\langle z \rangle$ with $\mathcal{E}_z = \{ z \}$. Note that $\mathcal{E} = \mathcal{E}_z$.

**Example 2. Simple case model.** Figure 2 shows a model with a single case element $w$ containing one case file item $o$, a connector $p$, and a task $q$. An event from case file item $o$ triggers the entry criteria 'k' that will allow task $q$ to execute. The optional connector $p$ visualizes the event propagation between case file item $o$ and the entry criteria 'k' of task $q$. The model has four model elements $\mathcal{E} = \{ w, o, p, q \}$, one scope element $\mathcal{M} = \{ w \}$, and one case element $\mathcal{L} = \{ w \}$. The scope relationship is $\mathcal{U} = \{ [o, p], [o, q], [p, q] \}$. The event relationship is $\mathcal{V} = \{ [o, q] \}$. The annotator set is $\mathcal{A} = \{ k, j, l \}$. This example contains one module $\langle w \rangle$ with $\mathcal{E}_w = \{ w, o, p, q \}$. Note that $\mathcal{E} = \mathcal{E}_w$.

**Example 3. Model with two cases.** Figure 3 shows a model with two case elements $x$ and $z$. This model con-
The annotator set contains three scope elements $M = \{x, y, z\}$, and two case elements $L = \{x, z\}$. The model has eight modeling elements $E = \{a, b, c, d, e, x, y, z\}$. There are three scopes, $x$ with $\{a, b, c, y\}$, $y$ with $\{d, e\}$, and one empty scope $z$. The scope relationship $U = \{[a, c], [a, b], [a, y], [b, c], [b, y], [c, y], [d, e]\}$. Note that $\{a, c\}$ and $\{d, e\}$ are in different scope, and so, $[a, d] \notin U$. The same is true for $[a, c], [b, d], [b, e], [c, d], [c, e]$ that are not in $U$. The event relationship $V = \{(a, c)\}$. The annotator set $A = \{i, j, k, l, m\}$. This example contains three modules, $\langle x \rangle$ with $E_x = \{x, a, b, c, y, d, e\}$, $\langle y \rangle$ with $E_y = \{y, d, e\}$, and $\langle z \rangle$ with $E_z = \{z\}$. Note that by (3.5) $L = \{x, z\} \implies E = E_x \cup E_z$. Figure 4 shows a tree view of the modeling elements $E$ in this example. The tree view is for illustration purposes and it is not a CMMN diagram.

3.3 Metrics

We define three metrics that are consistent with the formal framework for software measurements defined by Briand et al. [4]. Although we are interested in complexity metrics, we also define size and length, because Muletha et al. [23] concluded that size and length are similar to the complexity activity metrics proposed by Cardoso [6].

**Definition 7. (Size metric):** The size of a model $C$ denoted by $CS(C)$ is defined as the cardinality of $E$,

$$CS(C) = |E| \quad (3.8)$$

The size of a module $\langle z \rangle$ is defined as the cardinality of $E_z$,

$$CS(\langle z \rangle) = |E_z|$$

By (3.5) it follows that,

$$CS(C) = |E| = \sum_{x \in L} |E_x| = \sum_{x \in L} CS(\langle x \rangle)$$

Note that,

- Size for Figure 1 is $CS(\text{example1}) = |E| = 1$.
- Size for Figure 2 is $CS(\text{example2}) = |E| = 4$.
- Size for Figure 3 is $CS(\text{example3}) = |E| = 8$.
- Size for $\langle x \rangle$ in Figure 3 is $CS(\langle x \rangle) = |E_x| = 7$.
- Size for $\langle y \rangle$ in Figure 3 is $CS(\langle y \rangle) = |E_y| = 3$.
- Size for $\langle z \rangle$ in Figure 3 is $CS(\langle z \rangle) = |E_z| = 2$.

**Definition 8. (Length metric):** The length of a model $C$ denoted by $CL(C)$ is defined as the maximum nesting depth of a module. The length $CL(C)$ can be calculated by the following algorithm,

```python
int func CL(C) {
    int m := 0
    for each case-plan $c \in L$ do
        $m := \max(m, \text{depth}(m, c))$
    od
    return $m$
}
```

The length of a module $\langle z \rangle$ is defined as the maximum nesting depth of the module, and can be calculated using the following algorithm,

```python
int func depth(s) {
    int d := 0
    if ($s \notin M$) then return 0 fi
    for all c in scope s do
        $d := \max(d, \text{depth}(c))$
    od
    return $d + 1$
}
```

As shown in Figure 4, the elements $E$ of a model $C$ can be organized as a forest graph, where each tree is a case element module. We used the CMMN element icons to represent the nodes of the tree. Figure 4 shows a forest with two trees $\langle x \rangle$, $\langle z \rangle$, and a subtree $\langle y \rangle$. It also shows six leaves, namely $d$, $e$, $a$, $b$, $c$, and $z$.

Note that,

- Length for Figure 1 is $CL(\text{example1}) = 1$.
- Length for Figure 2 is $CL(\text{example2}) = 2$.
- Length for Figure 3 is $CL(\text{example3}) = 3$.
- Length for $\langle x \rangle$ in Figure 3 is $CL(\langle x \rangle) = 3$.
- Length for $\langle y \rangle$ in Figure 3 is $CL(\langle y \rangle) = 2$.
- Length for $\langle z \rangle$ in Figure 3 is $CL(\langle z \rangle) = 1$.

**Definition 9. (Complexity metric):** The complexity of a model $C$ denoted by $CC(C)$ is defined as,

$$CC(\emptyset) = 0, \text{ otherwise } CC(C) = \sum_{i \in E \cup A} W_i \quad (3.9)$$

Where, the weight, $W_i$ is given in Table 3.

The complexity of a model resembles a cognitive complexity metric with cognitive weights. There are several cognitive complexity metrics defined for BPM [29, 10, 11, 7]. However, those cognitive complexity metrics are not applicable to CMMN, because they are based on control structures [9] like sequence, branching, iterations, etc., which are
not present in CMMN. For the CMMN complexity metric
$CC(C)$, instead we assign weights to elements in $E$ and annotators in $A$. The weights were assigned based on the authors’ intuition. Higher weight were given to less frequently used elements and to annotators that increase the number of elements in the visual canvas, because less frequently used elements require better recall capabilities by users and clutter by annotators makes the model harder to read.

The complexity of a module $⌜z⌝$ is defined as,

$$CC(⌜z⌝) = \sum_{i \in E_1 \cup A_1} W_i$$

Note that,

Complexity for Figure 1 is $CC(⌜example1⌝) = 1$.
Complexity for Figure 2 is $CC(⌜example2⌝) = 7$ (weights $w=1$, $o=1$, $p=0$, $q=1$, $k=1$, $l=0$, $j=3$).
Complexity for Figure 3 is $CC(⌜example3⌝) = 11$ (weights $z=1$, $x=2$, $b=0$, $c=1$, $y=1$, $d=1$, $e=1$, $h=1$, $i=0$, $j=0$, $k=1$, $l=0$, $m=1$).
Complexity for $⌜x⌝$ in Figure 3 is $CC(⌜x⌝) = 10$ (weights $x=1$, $a=2$, $b=0$, $c=1$, $y=1$, $d=1$, $e=1$, $h=1$, $i=0$, $j=0$, $k=1$, $l=0$, $m=1$).
Complexity for $⌜y⌝$ in Figure 3 is $CC(⌜y⌝) = 3$ (weights $y=1$, $d=1$, $e=1$).
Complexity for $⌜z⌝$ in Figure 3 is $CC(⌜z⌝) = 1$.

### 4. THEORETICAL VALIDATION

In this section a theoretical validation of each metric is conducted using the Briand et al. framework [3]. The complexity metric is further validated against the nine properties for software complexity measures defined by Weyuker [36].

#### 4.1 Briand’s framework

The Briand et al. framework [4] categorizes software metrics into size, length, complexity, cohesion, and coupling. The framework is based on systems and modules. Briand et al. defines a system $S$ as a pair $S = (E, R)$, where $E$ represents a set of elements of $S$, and $R$ is a binary relationship on $E$ $(R \subseteq E \times E)$. A module of $S$ is defined as $m = (E_m, R_m)$ for $E_m \subseteq E$, $R_m \subseteq E_m \times E_m$ and $R_m \subseteq R$.

For our purposes a system corresponds to a model $C$ and a module corresponds to a module $(⌜m⌝, m \in M)$. $E$ corresponds to modeling elements $E (E = \mathcal{E})$ as described in Table 1. $R$ corresponds to the two binary relationships in $C$ ($U$ and $V$). We define $R$ as follows,

$$R = \{[a, b] \mid [a, b] \in U\} \cup \{(a, b) \mid (a, b) \in V\}$$

Note that $[a, b] \neq (a, b)$, $[a, b]$ is an unordered pair, while $(a, b)$ is an ordered pair.

<table>
<thead>
<tr>
<th>$\mathcal{E} \cup A$</th>
<th>Description</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>case element</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>stage element</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>discretionary stage element</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>plan fragment element</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>case file item element</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>task element</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>discretionary task element</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>event listener element</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>milestone element</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>connector (sentry) element</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>collapsed planing table</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>expanded planing table</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>auto complete</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>collapsed</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>expanded</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>manual activation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>repetition</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>entry criterion with associated connector</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>entry criterion without a connector</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>exit criterion with associated connector</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>exit criterion without a connector</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>non-blocking human</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>process</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>case referring to a case element not in this model</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>case referring to a case element in this model</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>participant</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>timer</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3: CMMN weights
We will use the terminology used in the Briand et al. framework [4] to introduce each property followed by a short proof against our metrics.

### 4.1.1 Size

Briand et al. define a function \( \text{Size}(S) \) characterized by three properties.

**Size 1. Non-negativity.** The size of a model \( S = (E, R) \) is non-negative.

\[
(S = (E, R)) \implies \text{Size}(S) \geq 0
\]

**Proof:** By definition 7, \( \text{Size} \) is the cardinality of \( E \) and the cardinality of a set cannot be negative.

\[
\therefore \quad \text{CS}(C) = |E| \geq 0
\]

**Size 2. Null value.** The size of a model \( S = (E, R) \) is zero if \( E \) is empty.

\[
(S = (E, R) \land E = \emptyset) \implies \text{Size}(S) = 0
\]

**Proof:** By definition, the cardinality of an empty set is zero.

\[
\therefore \quad \text{CS}(C) = |E| = |\emptyset| = 0
\]

**Size 3. Module additivity.** The size of a module \( S = (E, R) \) is equal to the sum of the sizes of two of its modules \( m_1 = (E_{m_1}, R_{m_1}) \) and \( m_2 = (E_{m_2}, R_{m_2}) \) such that any element of \( S \) is in either \( m_1 \) or \( m_2 \).

\[
(m_1 \subseteq S \land m_2, E = E_{m_1} \cup E_{m_2} \land E_{m_1} \cap E_{m_2} = \emptyset) \implies \text{Size}(S) = \text{Size}(m_1) + \text{Size}(m_2)
\]

**Proof:** Consider a model \( C \) with two disjoint modules \( \langle x \rangle \) and \( \langle y \rangle \), such that each element in \( C \) is either in \( \langle x \rangle \) or in \( \langle y \rangle \), but not both \( \langle E_x \cap E_y = \emptyset \rangle \) \( \langle E_x \cup E_y \rangle \). It follows, based on (3.5), that \( x \) and \( y \) are the only two case elements in \( E \{L = \langle x, y \rangle \} \), thus \( E \) is partitioned by \( \langle x \rangle \) and \( \langle y \rangle \).

\[
\therefore \quad \text{CS}(C) = |E| = |E_x| + |E_y| = \text{CS}(\langle x \rangle) + \text{CS}(\langle y \rangle)
\]

### 4.1.2 Length

Briand et al. define a function \( \text{Length}(S) \) characterized by five properties.

**Length 1. Non-negativity.** The length of a model \( S = (E, R) \) is non-negative.

\[
(S = (E, R)) \implies \text{Length}(S) \geq 0
\]

**Proof:** \( \text{CL} \) is defined as the maximum nesting depth of a model, and calculated with algorithm \( \text{CL}(C) \). Analyzing algorithm \( \text{CL}(C) \), the variables (\( m \) and \( d \)) are initialized to zero, and only increased by one or assigned the maximum of itself and depth which always returns \( d + 1 \).

\[
\therefore \quad \text{CL}(C) \geq 0
\]

**Length 2. Null value.** The length of a model \( S = (E, R) \) is zero if \( E \) is empty.

\[
(S = (E, R) \land E = \emptyset) \implies \text{Length}(S) = 0
\]

**Proof:** Consider an empty model \( C \), then \( E = \emptyset \implies L = \emptyset \) (because \( L \subseteq E \)). Analyzing algorithm \( \text{CL}(C) \), it initializes \( m \) to zero, and if \( L = \emptyset \) then ‘depth(m, c)’ is never invoked, forcing the return of \( m \) which is zero.

\[
\therefore \quad (E = \emptyset) \implies \text{CL}(C) = 0
\]

**Length 3. Non-increasing monotonicity for connected components.** Adding relationships between elements of a module \( m \) does not increase the length of the model \( S = (E, R) \).

\[
(S = (E, R) \land m = (E_{m_1}, R_{m_1}) \land m \subseteq S \land m \text{ is a connected component of } S \land S' = (E, R') \land R' = R \cup \{(e_1, e_2) \} \implies \text{Length}(S) \geq \text{Length}(S')
\]

**Proof:** Consider a model \( C \) with two modeling elements in a module \( \langle a, b \rangle \in \langle x \rangle \). There are two cases,

**case 1** \( a \) and \( b \) are in different scope. They cannot be related by scope, as that will require moving them within submodules of module \( \langle x \rangle \) adding \( \langle a, b \rangle \) to \( R \) which will change the structure of modules violating \( \langle e_1, e_2 \rangle \notin R \). They cannot be related by an event, because by definition 6 to be related by an event they are necessarily in the same scope.

**case 2** \( a \) and \( b \) are in the same scope. They are already related by scope \( \langle a, b \rangle \notin U \). Assuming \( \langle a, b \rangle \notin V \), we can relate them by an event and add them to \( V_m \), which corresponds to adding \( \langle a, b \rangle \) to \( R' \) and leaving \( R \) invariant.

Therefore, \( a \) and \( b \) can be related only by an event, and the scope relationship \( U \) is not affected. Then, neither the length of \( \langle x \rangle (\text{CL}(\langle x \rangle)) \), nor the length of the model \( C (\text{CL}(C)) \) has changed.

The maximum nesting depth of a forest is greater than or equal to the maximum nesting depth of any of the trees or subtrees, and \( \langle x \rangle \) is either a tree or a subtree on model \( C \).

\[
\therefore \quad \text{CL}(C) = \text{CL}(\langle x \rangle)
\]

**Length 4. Non-decreasing monotonicity for non-connected components.** Adding relationships between the elements of two modules \( m_1 \) and \( m_2 \) does not decrease the length of the model \( S = (E, R) \).

\[
(S = (E, R) \land m_1 = (E_{m_1}, R_{m_1}) \land m_2 = (E_{m_2}, R_{m_2}) \land m_1 \subseteq S \land m_2 \subseteq S \land m_1, m_2 \text{ are separate connected components of } S \land S' = (E, R') \land R' = R \cup \{(e_1, e_2) \} \land \langle e_1, e_2 \rangle \notin R \land e_1 \in E_{m_1} \land e_2 \in E_{m_2} \implies \text{Length}(S) \geq \text{Length}(S')
\]

**Proof:** Adding modules. The length of a model \( S = (E, R) \) made up of two disjoint modules \( m_1 \) and \( m_2 \) is equal to the maximum of the lengths of the modules \( m_1 \) and \( m_2 \).

\[
(S = m_1 \cup m_2 \land m_1 \cap m_2 = \emptyset \land E = E_{m_1} \cup E_{m_2}) \implies \text{Length}(S) = \max(\text{Length}(m_1), \text{Length}(m_2))
\]

**Proof:** Consider a model \( C \) with two disjoint modules \( \langle x \rangle \) and \( \langle y \rangle \), such that \( E = E_x \cup E_y \land E_x \cup E_y = \emptyset \). Therefore, by (3.5) \( x \) and \( y \) are the only two case elements in \( E \{L = \langle x, y \rangle \} \) and they create the only two trees \( \langle x \rangle \land \langle y \rangle \) in the forest. The maximum nesting depth of a forest is equal to the maximum nesting depth of its trees.

\[
\therefore \quad \text{CL}(C) = \max(\text{CL}(\langle x \rangle), \text{CL}(\langle y \rangle))
\]
4.1.3 Complexity

Briand et al. define a function \( \text{Complexity}(S) \) characterized by five properties. Complexity for Briand et al. is distinct from cognitive complexity, as they state that complexity in the framework is an intrinsic attribute of an object and not a perceived psychological complexity.

**Complexity 1. Non-negativity.** The complexity of a model \( S = (E, R) \) must be non-negative.

\[
(S = (E, R)) \implies \text{Complexity}(S) \geq 0
\]

**Proof:** By definition 9, \( \text{CC}(C) = \sum_{i \in E \cup A} W_i \), where weight \( W_i \) is a positive integer from 0 to 3 given by Table 3. Suppose we replace each element in \( E \) for its weight and call the resulting set \( E' \), and each annotator in \( A \) for its weight and call that set \( A' \). Then, \( \text{CC}(C) = \sum_{i \in E \cup A} W_i = \sum_{p \in E' \cup A'} p \) and \( p \in N_0 \), therefore \( (\sum_{p \in E' \cup A'} p) \in N_0 \).

\[
\therefore \text{CC}(C) \geq 0.
\]

**Complexity 2. Null value.** The complexity of a model \( S = (E, R) \) is zero if \( R \) is empty.

\[
(S = (E, R) \land R = \emptyset) \implies \text{Complexity}(S) = 0
\]

**Proof:** By definition 9, the complexity of an empty model is zero.

\[
\therefore (C = \emptyset) \implies R = U = V = \emptyset \land \text{CC}(C) = 0
\]

**Complexity 3. Symmetry.** The complexity of a model \( S = (E, R) \) does not depend on the convention chosen to represent the relationships between its elements.

\[
(S = (E, R) \land S^{-1} = (E, R^{-1})) \implies \text{Complexity}(S) = \text{Complexity}(S^{-1})
\]

**Proof:** For a model \( C, R \) is given by the two relationships \( U \) and \( V \). By definition 9, \( \text{CC}(C) = \sum_{i \in E \cup A} W_i \), which does not depend on \( U \) or \( V \), or the convention used to represent \( U \) and \( V \).

\[
\therefore \text{CC}(C) = \text{CC}(C')
\]

**Complexity 4. Module monotonicity.** The complexity of a model \( S = (E, R) \) is no less than the sum of the complexities of any two of its modules with no relationships in common.

\[
(S = (E, R) \land m_1 = (E_{m_1}, R_{m_1}) \land m_2 = (E_{m_2}, R_{m_2}) \land m_1 \cup m_2 \subseteq S \land R_{m_1} \cap R_{m_2} = \emptyset) \implies \text{Complexity}(S) \geq \text{Complexity}(m_1) + \text{Complexity}(m_2)
\]

**Proof:** Consider a model \( C \) with two modules \( m_1 \) and \( m_2 \) such that \( E_{m_1} \cup E_{m_2} \subseteq E \land E_{m_1} \cap E_{m_2} = \emptyset \). We can ignore relationship \( R = U \cup V \), because by definition 9, \( \text{CC}(C) \) does not depend on \( U \) or \( V \).

Because \( E_{m_1} \cup E_{m_2} \subseteq E \), we can define \( T \) such that \( E = T \cup E_{m_1} \cup E_{m_2} \land T \cap E_{m_1} = T \cap E_{m_2} = T \cap E_{m_1} \cap E_{m_2} = \emptyset \).

Annotators in \( A \) are associated with elements in \( E \), therefore we can define a function \( f \) such that \( f : A \to E \). We can separate the elements of \( A \) such that \( A_{m_1} = \{ x \mid f(x) \in E_{m_1} \}, A_{m_2} = \{ x \mid f(x) \in E_{m_2} \}, \) and \( A_T = \{ x \mid f(x) \in T \} \).

By way of contradiction, assume \( A_{m_1} \cap A_{m_2} \cap A_T = \{ \} \).

That means there is an annotator \( a \) in \( A \) for which \( f(a) \notin T \cup E_{m_1} \cup E_{m_2} \), but \( E = T \cup E_{m_1} \cup E_{m_2} \). We have an annotator \( a \) without an image in \( E \) which contradicts our definition of \( f \).

\[
\therefore \text{CC}(C) = \text{CC}(m_1) + \text{CC}(m_2)
\]

4.2 Weyuker’s properties

Weyuker [36] proposed a set of nine properties of complexity software metrics that have been widely used to validate business process metrics [23]. Briand et al. [4] found that the Weyuker [36] properties are consistent with complexity in his framework.

The Weyuker [36] notation uses \( P, Q, \) and \( R \) to denote programs, which for our purposes corresponds to CMMN models. To avoid confusion with the Briand et al. [4] framework, we will use \( T \) instead of \( R \). Weyuker uses the set cardinality operator \( |P| \) to denote complexity, which we already used to denote cardinality. Therefore, we will use \( ||P|| \) as the complexity of model \( P \). Weyuker expects that for any \( P \), its complexity \( ||P|| \) is a nonnegative number, hence complexity can be compared and ordered.

\[
||P|| \leq |Q| \lor |Q| \leq ||P||
\]

We will use the terminology used by Weyuker [36] and the classification by Srinivasan and Devi [30] to introduce each property, followed by a short proof against our complexity metric \( CC \).

**Property 1. Non-coarseness.** A metric should not rank all models as equally complex.

\[ \exists P \exists Q \left( ||P|| \neq ||Q|| \right) \]
Proof: Let $P$ be example 2 (see Figure 2) with $CC(P) = 7$, and $Q$ be example 3 (see Figure 3) with $CC(Q) = 11$.

$$\therefore \quad (\exists P)(\exists Q)(CC(P) \neq CC(Q))$$

Property 2. Granularity. A metric should rank only a finite number of models with the same complexity.

Let $c$ be a non-negative number, then there are only finitely many models of complexity $c$.

Proof: Assuming a model can only be renamed (see property 8) in a finite number of ways, then consider a number $c \in \mathbb{N}$, and a model $P$ such that $CC(P) = c$. By definition 9, $CC(P)$ is calculated using $\mathcal{E}_P$ and $\mathcal{A}_P$. We need to show there is a finite number of $\mathcal{E}_P \cup \mathcal{A}_P$ sets such that $CC(P) = c$. Note that if $c = 0$ then $\mathcal{E}_P \cup \mathcal{A}_P = \emptyset \land CC(\emptyset)$, thus there is only one set $\emptyset \cup \emptyset$ that gives $c = 0$.\qed

We start by proving that $\mathcal{E}_P \cup \mathcal{A}_P$ is finite for $c > 0$. The only element in $\mathcal{E}_P$ (see Table 1) that has a weight of 0 is the connector (see Table 3), but a connector must be associated with a send in $\mathcal{A}_P$ with weight of 1 (entry or exit criteria). We cannot add an infinite number of connectors to $\mathcal{E}_P$ without adding sentries and changing the value of $CC(P)$. Thus, $CC(P) = c \implies |\mathcal{E}_P| \neq c$, and model $P$ has a size $CS(P) = |\mathcal{E}_P| = n$. Set $\mathcal{E}_P$ is finite ($\exists n(|\mathcal{E}_P| \leq n)$. By definition 1, each element in $\mathcal{A}_P$ is associated with a single element in $\mathcal{E}_P$. In $\mathcal{A}_P$ (see Table 2) markers and decorators are bound to $|\mathcal{E}_P|$, but sentries are not. However, sentries (entry and exit criterion) weights range from 1 to 3. We cannot have an infinite numbers of sentries, because the number is bound by $c$ (i.e. $\sum_{i \in \{1,2,3\} \times |\mathcal{A}_P|} i \leq c$). Therefore, $\mathcal{A}_P$ is also finite and $(\exists n)(|\mathcal{A}_P| \leq m)$. Moreover $\mathcal{E}_P \cup \mathcal{A}_P = \emptyset \implies |\mathcal{E}_P| + |\mathcal{A}_P| = |\mathcal{E}_P \cup \mathcal{A}_P| \leq n + m$. Therefore, $\mathcal{E}_P \cup \mathcal{A}_P$ is a finite set.

Without loss of generality, we can summarize Table 3 into four weight categories (0, 1, 2, 3), and we know $|\mathcal{E}_P \cup \mathcal{A}_P| \leq n + m$. Using brute force, we can count all combinations with repetition of four categories (0, 1, 2, 3) into 1 to $n + m$ slots, using $C(n + r - 1, r)$. That will give us all possible sets $\mathcal{E}_P \cup \mathcal{A}_P$ that can produce, among other complexities, $CC(P) = c$. We calculate $\sum_{r=1}^{r=n+m} C(4+r-1, r) = 2^{4+n+m-1}$. Therefore, there is a finite number of sets $\mathcal{E}_P \cup \mathcal{A}_P$ such that $CC(P) = \sum_{i \in \mathcal{E}_P \cup \mathcal{A}_P} W_i = c$.\qed

Property 3. Non-uniqueness (Notion of equivalence). A metric should allow some models to have the same complexity.

$(\exists P)(\exists Q)(P \neq Q \land ||P|| = ||Q||)$

Proof: Let $P$ be example 2 (see Figure 2) with $CC(P) = 7$, and $Q$ a similar model, but changing modeling element $o$ (case file item) to a task $t$. Thus, $Q$ is different from $P$ by one modeling element. The complexity of $Q$, namely, $CC(Q) = 7$, because the weight for a task is the same as the weight for a case file item (see Table 3).

$$\therefore \quad (\exists P)(\exists Q)(P \neq Q \land CC(P) = CC(Q))$$

Property 4. Design details are important. Two distinct but equivalent models that compute the same function need not have the same complexity.

$(\exists P)(\exists Q)(P \equiv Q \land ||P|| \neq ||Q||)$

Proof: Let $P$ be example 3 (see Figure 3) with $CC(P) = 11$, and $Q$ a similar model, but includes an extra task $t$ inside case $x$. Task $t$ is a dummy task that does nothing when it executes (a ‘skip’ statement). Operationally, $Q$ is equivalent to $P \equiv P$, because they compute the same function. However, the complexity of $Q$ is $CC(Q) = 12$, because task $t$ adds a weight of 1 (see Table 3).

$$\therefore \quad (\exists P)(\exists Q)(P \equiv P \land CC(P) \neq CC(Q))$$

Property 5. Monotonicity. The complexity of two models joined together is greater than or equal to the complexity of either model considered separate.

$(\forall P)(\forall Q)(||P|| \leq ||Q; P|| \land ||Q|| \leq ||P; Q||)$

Proof: Let $CC(P) = \sum_{i \in \mathcal{E}_P \cup \mathcal{A}_P} W_i$, and $CC(Q) = \sum_{i \in \mathcal{E}_Q \cup \mathcal{A}_Q} W_i$. Then, $CC(P; Q) = \sum_{i \in \mathcal{E}_P \cup \mathcal{E}_Q \cup \mathcal{A}_P \cup \mathcal{A}_Q} W_i = \sum_{i \in \mathcal{E}_Q \cup \mathcal{A}_Q} W_i = CC(Q; P)$. We have two cases,\newline

case 1 $P = \emptyset \lor Q = \emptyset$. Assume $P$ is an empty model, then $\mathcal{E}_P = \mathcal{A}_P = \emptyset$. Therefore, $CC(P; Q) = \sum_{i \in \mathcal{E}_Q \cup \mathcal{A}_Q} W_i = \sum_{i \in \mathcal{E}_Q \cup \mathcal{A}_Q} W_i = CC(Q)$. Assuming $Q$ is an empty model gives the same result $CC(P; Q) = CC(P)$. Assuming both $P$ and $Q$ are empty leads to the same result $CC(P; Q) = CC(P) = CC(Q) = 0$.

case 2 $P \neq \emptyset \land Q \neq \emptyset$. Then, $CC(P; Q) = \sum_{i \in \mathcal{E}_P \cup \mathcal{E}_Q \cup \mathcal{A}_P \cup \mathcal{A}_Q} W_i = \sum_{i \in \mathcal{E}_Q \cup \mathcal{A}_Q} W_i + \sum_{i \in \mathcal{E}_Q \cup \mathcal{A}_Q} W_i = CC(Q) + CC(P)$.

$$\therefore \quad (\forall P)(\forall Q)(CC(P) \leq CC(P) + CC(Q) \land CC(Q) \leq CC(P) + CC(Q))$$

Property 6. Nonequivalence of interaction. Two models with the same complexity when each is joined to a third model the resulting complexity may be different between the two.

a.$(\exists P)(\exists Q)(\exists T)(||P|| = ||Q|| \land ||P; T|| \neq ||Q; T||)$

b.$(\exists P)(\exists Q)(\exists T)(||P|| = ||Q|| \land ||T; P|| \neq ||T; Q||)$

Proof: For complexity $CC$ the order of concatenation of models is not important. Let $CC(P) = \sum_{i \in \mathcal{E}_P \cup \mathcal{A}_P} W_i$, and $CC(T) = \sum_{i \in \mathcal{E}_T \cup \mathcal{A}_T} W_i$. Then, $CC(P; T) = \sum_{i \in \mathcal{E}_P \cup \mathcal{E}_T \cup \mathcal{A}_P \cup \mathcal{A}_T} W_i = CC(T; P)$ Therefore, $CC(P; T) = CC(T; P)$ which is the same as $||P; T|| = ||T; P||$, and we have only one case.

Let $P$ be example 2 (see Figure 2) with $CC(P) = 7$, and $Q$ similar to $P$ but changing modeling element $o$ to a task $t$. Thus, $Q$ is different from $P$ by one modeling element, still $CC(P) = CC(Q) = 7$ (same as in property 3). Now let the case annotator ‘l’ in $P$ invoke case $z$, and case annotator ‘l’ in $Q$ invoke case $x$. That does not change $CC(P)$ or $CC(Q)$, because neither case $z$ or case $x$ are in the models, therefore the weight of ‘l’ remains 0 (see Table 3).

Let $T$ be example 1 (see Figure 1) with $CC(T) = 1$. Concatenating $P$ with $T$ produces $CC(P; T) = 9$ (weights $z=1$, $w=1$, $o=1$, $p=0$, $q=1$, $k=1$, $l=1$, $j=3$), where ‘l=1’, because case $z$ is in $P$; $T$. However, concatenating $Q$ with $T$ is $CC(Q; T) = 8$ (weights $z=1$, $w=1$, $o=1$, $p=0$, $q=1$, $k=1$, $l=0$, $j=3$), where ‘l=0’, because case $x$ is not in $Q; T$.

$$\therefore \quad (\exists P)(\exists Q)(\exists T)(CC(P) = CC(Q) \land CC(P; T) \neq CC(Q; T))$$
Property 7. Permutation. Complexity should be responsive to the order of statements.

$(\exists P)(\exists Q)(\text{Permutation}(Q, P) \land \|P\| = \|Q\|)$

Proof: Let $P$ be example 2 (see Figure 2) with $CC(P) = 7$, and $Q$ be similar to $P$ but with connector $p$ attached to exit criteria 'j' instead of entry criteria 'k'. Thus, $Q$ is a permutation of $P$ with $CC(Q) = 6$, because in $CC(Q)$ the weight of 'k' increased from 1 to 2, and the weight of 'j' decreased from 3 to 1 (see Table 3).

$\therefore (\exists P)(\exists Q)(\text{Permutation}(Q, P) \land CC(P) \neq CC(Q))$

Property 8. Renaming. Complexity should not be affected by renaming.

$(\forall P)(\forall Q)(\text{Rename}(Q, P) \land \|P\| = \|Q\|)$

Proof: Let $P$ be example 2 (see Figure 2) with $CC(P) = 7$, and $Q$ be similar to $P$ but with different names. Suppose we rename $w, a, p, q, k, l, j \in E_P$ to $a, b, c, d, e, f, j, q$ in $Q$. Thus, $Q$ is a renaming of $P$. Note that $Q$ has the same number and type of modeling elements and annotators as $P$, and they are organized in exactly the same way. Thus, the complexity of $Q$ remains $CC(Q) = 7$. This can be done with any model.

$\therefore (\forall P)(\forall Q)(\text{Rename}(Q, P) \land CC(P) = CC(Q))$

Property 9. Interaction may increase complexity.

$(\exists P)(\exists Q)(\|P\| + \|Q\| < \|P; Q\|)$

Proof: Let $P$ be example 1 (see Figure 1) with $CC(P) = 1$, and $Q$ be example 2 (see Figure 2) with $CC(Q) = 7$. Assume that case annotator 'i' in $Q$ invokes case $z$, then in $Q$ the weight of case annotator 'j' is 0, but when joined with $P$ that has case $z$, the weight of annotator 'i' becomes 1, and $CC(P; Q) = 9$ (see Table 3).

$\therefore (\exists P)(\exists Q)(CC(P) + CC(Q) < CC(P; Q))$

5. FINDINGS AND FUTURE RESEARCH

The three proposed metrics comply with the formal framework for software measurements defined by Briand et al. [4], and the proposed complexity metric also complies with the properties described by Weyuker [36]. However, both Briand et al., and Weyuker assume that software systems are build using a procedural style, based on directed acyclic graphs. Briand et al. use directed acyclic graphs to describe their framework and to provide examples. Weyuker uses a procedural language to illustrate her properties. Therefore, work is required to understand the applicability of Briand et al., and Weyuker to declarative systems.

The validation of complexity metrics for business processes requires both theoretical and empirical validation [23, 22, 30]. Work is required to conduct the empirical validation for the proposed metrics. Further work on empirical validation using suggestions from Misra et al. [22] and Kitchenham et al. [16] should be conducted.

The formalization of the CMMN model was sufficient to define and validate the metrics. But, CMMN claims an approach based on business artifacts [19], therefore further work is required to compare the formal CMMN model described here with a formalization of business artifacts.

Further research is required on the complexity metric. The weights given for the complexity metric $CC(C)$ were assigned based on the intuition of the authors, and further empirical work is needed to fine tune the weights.

The CMMN specification defines non-visual entities that are not represented in the CMMN models (they are not described in Tables 1 or 2). Some of those entities are roles and non-visualized system events. Empirical work is needed to understand the influence of CMMN non-visual entities on the complexity metric.

6. CONCLUSIONS

This study provides a formal description of CMMN [25] and three metrics have been defined. The metrics were successfully validated using the formal framework for software measurements defined by Briand et al. [4], and the properties for software complexity metrics defined by Weyuker [36]. However, it is clear that both Briand et al. and Weyuker’s properties are designed for procedural models and not for declarative models. Therefore, further theoretical validation is required. In addition, further research is required to empirically validate the metrics and the complexity weights.

7. REFERENCES
