Meaningful Metrics for Multi-Level Modelling

Thomas Kühne
Victoria University of Wellington
Wellington, New Zealand
thomas.kuehne@ecs.vuw.ac.nz

Arne Lange
University of Mannheim
Mannheim, Germany
lange@informatik.uni-mannheim.de

ABSTRACT

One of the key enablers of further growth of multi-level modeling will be the development of objective ways to allow multi-level modeling approaches to be compared to one another and to two-level modeling approaches. While significant strides have been made regarding qualitative comparisons, there is currently no adequate way to quantitatively assess to what extent a multi-level model may be preferable over another model with respect to high-level qualities such as understandability, maintainability, and control capacity. In this paper, we propose *deep metrics*, as an approach to quantitatively measure high-level model concerns of multi-level models that are of interest to certain stakeholders. Beyond the stated goals, we see deep metrics as furthermore supporting the comparison of modeling styles and aiding modelers in making individual design decisions. We discuss what makes a metric "depth-aware" so that it can appropriately capture multi-level model properties, and present two concrete proposals for metrics that measure high-level multi-level model qualities.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**; • **Computing methodologies → Modeling methodologies**.

KEYWORDS

multi-level modeling, metrics, model comparison

ACM Reference Format:
Thomas Kühne and Arne Lange. 2020. Meaningful Metrics for Multi-Level Modelling. In *Proceedings of MULTI 2020: International Workshop on Multi-Level Modeling in conjunction with MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems Proceedings (MULTI 2020).* ACM, New York, NY, USA, 9 pages. https://doi.org//10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Multi-level modeling as a field has two ways to grow. First, it could increase the level of adoption by furthering its case through objective demonstrations of the superiority of multi-level models over two-level models. While many convincing qualitative arguments have been made [21] and deficiencies in using two-level approaches have been pointed out [9], the more persuasive approach of using quantitative comparisons has not been fully utilized yet. Gerbig used a metrics-based approach to compare a multi-level model to a two-level equivalent [15], but since he employed relatively basic two-level metrics, the comparison was not suited to measure high-level qualities in a manner that would adequately reflect the idiosyncrasies of a multi-level model.

The second way in which multi-level modeling can grow as a field is to support objective comparisons between various multi-level modeling approaches [4, 8]. Previous attempts at comparing multi-level languages with each other were based on language feature comparisons [2, 18] or employed so-called "Modeling Challenges" to qualitatively compare different multi-level modeling approaches through standardizing the domain and requirements [1, 11]. The latter have allowed to approach observing the concrete effect language choice may have on model properties such a complexity. However, while some intuitive evaluation criteria such as model size and model maintainability exist, no standardized definitions were put forward in these challenges and no attempt at achieving quantitative comparisons was made.

Metrics are a commonly accepted approach for supporting the objective evaluation of software artifacts such as programs or models [16]. While some low-level metrics such as "depth of inheritance" may not be directly meaningful in term of more high-level and complex qualities such as "maintainability", metrics overall have the potential to produce at least useful indicators for important model qualities. As long as a community can agree that particular metrics correlate with desirable model qualities, metrics can be an objective and inexpensive method to compare models to each other. Such comparisons could reveal differences between various multi-level modeling approaches, as their respective differences in choice of concepts will imply different models for the same domain scenario. Metric-based comparisons could furthermore reveal differences between multi-level vs two-level solutions, ideally accounting for both negative and positive aspects of using multiple levels instead of just two. Finally, metrics could be utilized to compare modeling styles to each other or even support modelers in making micro-design decisions.

In the following, we first clarify what exactly we want to target with our metrics in the context of multi-level modeling. Next we discuss the notion of *meaningful metrics*. Subsequently, we look at what *deep metrics* need to take into consideration in comparison to traditional metrics. We then briefly discuss two deep metrics candidates. Finally, we discuss related work, potential future work, and conclude.

MULTI 2020, October 16–23, 2020, Virtual

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$0.00
https://doi.org//10.1145/nnnnnnn.nnnnnnn

2 METRICS TARGETS

Prior to measuring something, it is important to define what exactly should be measured. Some traditional software metrics target qualities such as robustness, efficiency, availability, reliability, etc. which are clearly properties of the final product (e.g., deployed code), rather than properties of the source (e.g., a model or source code). Fig. 1 labels the target of respect metrics as "Product Quality". Some authors refer to such qualities as "external quality characteristics" [26]. Analyzing a model may support predictions of such product properties, but the properties are nevertheless properties of the product as opposed to properties of the model.

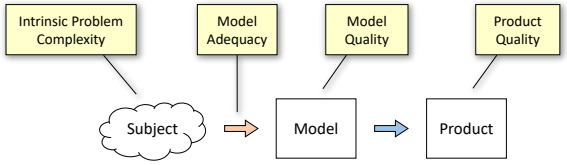

Figure 1: Potential Metrics Targets

It can also be useful to measure how adequate the model is with respect to its subject (cf. Fig. 1 "Model Adequacy"). Relevant questions include: Does the model –

- cover the subject with respect to its intended purpose?
- achieve sufficient fidelity with respect to the subject and the model's purpose?
- exhibit "direct mapping" [28]?
- add "accidental complexity" [7] to the "Intrinsic Problem Complexity" shown in Fig. 1?

All these questions can only be answered by considering both the subject of the model and the model in combination.

Finally, there are qualities associated with a model that can be assessed without reference to the model's subject and that are not directly associated with the product either. Model size, coupling between and cohesion of model elements (cf. [17]), understandability, and maintainability are examples for what in Fig. 1 is referred to as "Model Quality" and some authors refer to as "internal quality characteristics" [26].

In the following, we will focus on the latter model qualities. Examining product quality is out of scope since with respect to product properties it is (almost, see below) irrelevant what the model qualities of any supporting models are. Examining model adequacy, on the other hand, would very much be on-topic, in particular with respect to comparing multi-level models to two-level models. However, such investigations necessitate the formalization of the subject (domain of discourse) (cf. [19]). For the purposes of this paper, we therefore focus on model properties whose measurements only involve the model itself.

We note, however, that model qualities are

(1) influenced by model adequacy, and are
(2) expected to have an indirect impact on product quality.

Regarding (1), in particular, accidental complexity [7] is understood to impact on internal model qualities such as understandability and maintainability. As such, measuring model qualities could very well imply indirectly measuring, at least in part, model adequacy. Comparisons based on measuring models therefore either need to ensure that the models to be compared share the same level of adequacy, or indeed, could infer that different quality measurements are due to different levels of adequacy (e.g., in multi-level vs two-level comparisons).

Regarding (2), since internal qualities are generally acknowledged to be the key to achieving external qualities (here "Product Quality") [32], it is not only useful to initially focus on model properties for pragmatic purposes, but also in terms of overall relevance. Note that we will restrict our focus to metrics relating to model elements and their relationships. Considering the internals of operations (e.g., clabject methods) would open up the scope to include notions like McCabe's cyclomatic complexity [25], but are not of immediate interest here.

3 MEANINGFUL METRICS

There are many low-level metrics, such as "class count", "attribute count", "average subtyping depth", etc., that are trivial to obtain but have a very tenuous, if any, relationship to high-level qualities like understandability and maintainability. Even more complex metrics such as coupling [10], measure a technical property that is not necessarily of interest to stakeholders. Only if, for instance, coupling is reliably correlated to a high-level quality such as maintainability, should it be considered as contributing to the latter's measurement, and can then be regarded as being stakeholder-relevant.

Note that Sommerville refers to low-level metrics as measuring "internal quality attributes" and to stakeholder-relevant qualities as "external quality attributes" [32]. Since this use of the "internal" vs "external" dichotomy conflicts with the terminology of other authors such as McConnell [26], we refrain from using this terminology altogether and use the terms "low-level" vs "high-level" when it comes to characterizing the level of ambition a metric has with respect stakeholder view relevance.

Table 1 lists the stakeholder views we regard as relevant in the context of this paper. None of these could be adequately supported by low-level metrics such as "number of elements", "depth of specialization", etc. In order to be a *meaningful metric* to the respective stakeholders, a metric has to correlate with a high-level model quality. In other words, we essentially see a "stakeholder relevance" dimension running orthogonal to the target dimension shown in Fig. 1 – which spans a spectrum of very simple low-level measurements to complexly defined high-level metrics.

View	Variable	Outcome
language comparisons	language choice	demonstration of language superiority
multi- vs two-level	classification depth	multi-level pros/cons
modeling guidelines	modeling style	style trade-off analysis
model development	individual design decisions	design optimization

Table 1: Stakeholder Views

A spectrum of metrics with increasing level of importance to multi-level modeling could be defined as follows:

(1) **Two-level metric.** Classic, two-level metric.
(2) **Depth-unaware multi-level metric.** Classic metric applied to a multi-level model.
(3) **Low-level deep metric.** Low-level metric which recognizes classification depth in a multi-level model.
(4) **High-level deep metric.** High-level metric which incorporates classification depth in order to produce a measurement that correlates with a stakeholder view.

To date, to the best of our knowledge, only metrics up to level (2) have been used in multi-level modeling research. The level (3) metrics are relatively "low hanging fruit" since they amount to determining properties such as "maximum classification depth" that are analogous to similar metrics on specialization hierarchies. The metrics we are most interested in are the level (4) metrics.

4 DEEP METRICS

There are two aspects of multi-level models that traditional, object-oriented metrics do not have to consider: The presence of

(1) an unbounded number of classification levels [5], and
(2) deep characterization [6].

4.1 Classification Dimension

Standard object-oriented metrics do not need to consider deep classification chains. For instance, when analyzing dependencies between elements, the only elements that may be affected due to changes to their types are the bottom-level (O_0) objects. Therefore, in a two-level setting the only elements that may easily have an impact on types upon change are supertypes. Note that in this context, we are not considering the impact of changes that may occur through associations; this type of dependency constitutes its own impact category since it can be partially addressed through encapsulation. Rather, we are focusing on change impact that occurs through subtyping interfaces (or lack thereof) which intentionally offer weak protection only.

Subtypes and their supertypes often tightly interact through shared attributes and overridden operations, resulting in excellent code sharing and extensible systems, which is why such designs are considered to be an integral part of the object-oriented approach. However, even though natural and intended, such tight integration comes at the cost of having very little protection against ripple effects when changes occur anywhere within a subtype hierarchy branch. The strong mutual dependency between subtypes and their supertypes has lead authors to recommend *composition/forwarding* over inheritance [14]. With respect to operations, the respective lack of encapsulation between subtypes and supertypes leads to the so-called *fragile base class problem* [29]. Visibility scopes such as *private* in C++ or Java are intended to alleviate the fragility challenge but obviously cannot always help without entirely negating the advantages of subtyping. In summary, subtyping introduces some level of error-proneness and change dependency which we refer to as *subtyping-induced fragility*.

To illustrate the case of subtyping-induced fragility, consider Actor in Fig. 2 which is a supertype of two subtypes: SeniorActor and StdActor. When Actor was initially created, it embodied the requirement that all actors had to be available for eight hours per day. If subsequently the availability requirement for actors is relaxed from eight hours to four hours per day, e.g. to allow for part-time actors, and established at Actor, this change will have a ripple effect on SeniorActor. Senior actors are still required to be available for eight hours per day, so the respective requirement now has to be established at SeniorActor. If SeniorActor had any subtypes, then the fix to SeniorActor would restore the original requirement for these as well, illustrating that ripple effects are typically expected to die off near the "epicenter" of the change.

In multi-level models, types (at any level) may not only be derived from supertypes but may also be – and in fact are expected to be – derived from their own type at the level above. As a result, both the supertype and the type of a type (at any level) can determine the latter's shape in a multi-level model. Hence, a similar dependency to that between supertypes and subtypes exists in the form of *classification-induced fragility* for any ontologically typed element at any level in a multi-level model. Both sources of fragility can be viewed as being caused by a derivation dependency. We consider classification-induced fragility in Sect. 5.1 when proposing a metric for predicting the impact of change to a multi-level model.

4.2 Deep Characterization

The presence of metatypes (of any order) in multi-level models obviously not only has a detrimental impact on models. As a matter of fact, metatypes can reduce error-proneness when used to enforce certain structures at subjacent levels.

Metatypes (of any order) are used to their fullest potential when they exert *deep characterization*. For instance, a ProductType element at level O_2 can enforce the presence of a price slot for all product instances at level O_0, without depending on product types such as DVD to engage in a powertype scenario [6]. This means that any modeler who introduces a new product type to the system can never fail to make the new product instances have a price slot. Fig. 2 shows the deep metatype TaskType as performing an analogous role, ensuring that task enactments, such as PythonCoding, are guaranteed to have a duration slot. We consider this highly welcome increase of control when proposing a metric for capturing the increase of guarantees afforded by deep models in Sect. 5.2.

5 SPECIFIC METRIC PROPOSALS

In this section we propose two *deep metrics* whose validity still has to be established. Both are specifically designed to be applicable to deep models and are intended to provide meaningful measures of high-level model qualities.

5.1 Expected Change Impact

The "Expected Change Impact" (ECI) metric is intended to be used in contexts in which individual model elements are subject to change with a certain probability. The intent of the metric is to predict the expected impact of such changes, on average, including ripple effects. For now, we are only considering ripple effects caused by derivation dependencies (cf. Sect. 4.1), but the ECI metric should nevertheless correlate well with the notion of the *maintainability* of a model. The higher the expected overall change of impact, the

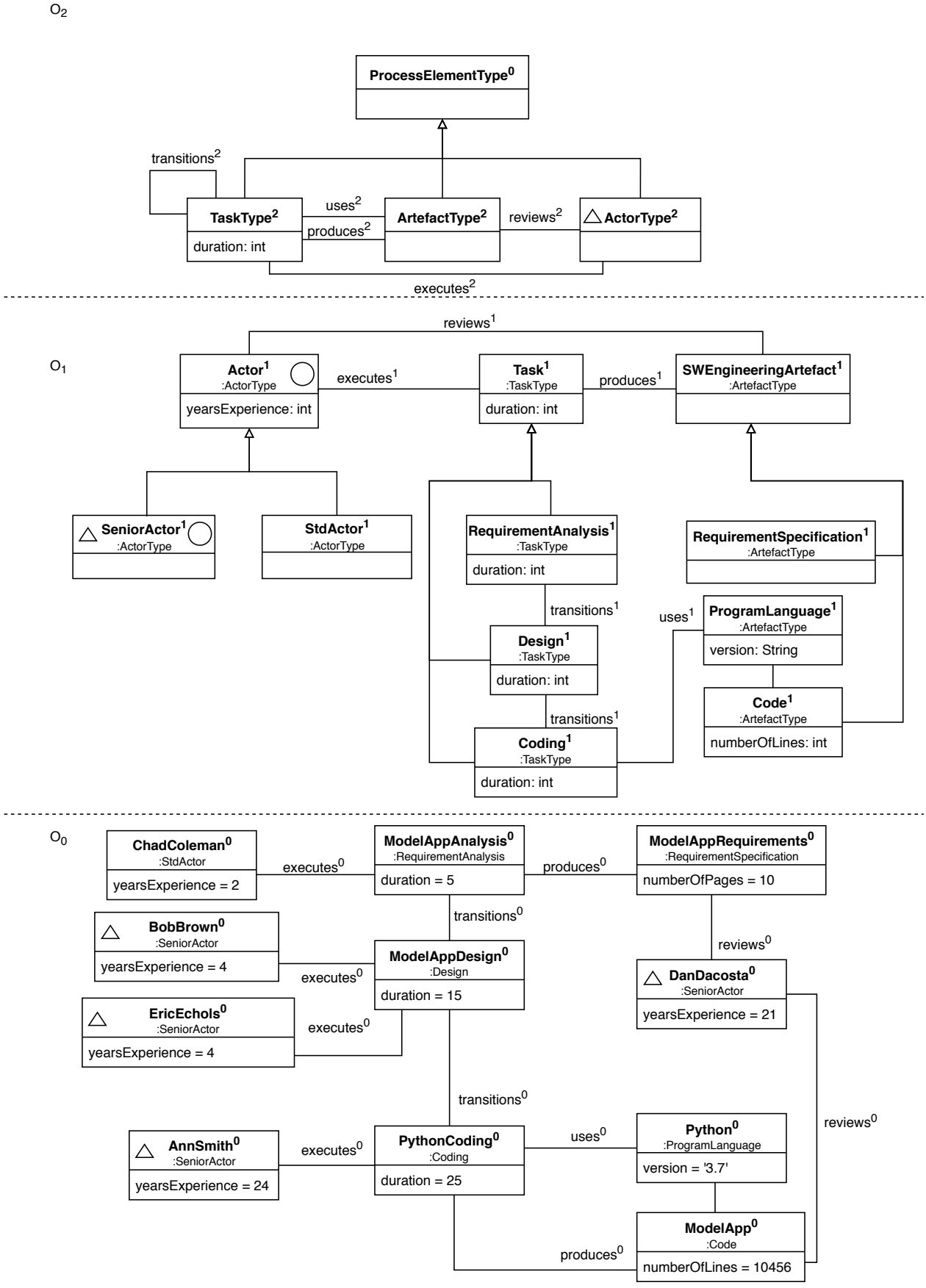

Figure 2: Multi-level model with applied metrics

more costly it is to maintain a model, assuming that all ripple effects must be attended to manually, incurring some labor cost.

In this context it is useful to define a local *impact scope* that comprises all elements that may potentially require attention after a particular element is changed. For instance, changing a type may affect the classification hierarchy upwards (i.e., its type and so on, recursively), the classification hierarchy downwards (i.e., its direct instances and so on, recursively), and analogously both upwards and downwards directions for the subtype hierarchy.

In order to maintain simplicity and since respective further contributions are expected to be negligible, for now we only consider the *first-order* impact scope of an element; Fig. 2 marks the elements in the first-order impact scope of SeniorActor, using triangles (classification) and circles (subtyping).

The ECI metric is intended to account for the following kinds of ripple effects:

- well-formedness repairs that become necessary within the impact scope of a changed element. For instance, moving an operation down from a supertype to a subtype may require the removal or change of operations in other subtypes that previously claimed to override a supertype operation. While such well-formedness repairs can in some cases be automated (e.g., adjusting a changed potency value in all derived elements [3]), not all can.
- model adequacy repairs that are required to maintain the model's level of fidelity in terms of representing the domain. For instance, changing the meaning of a class may imply a restructuring of all derived classes, whether they are subtypes or instances of the changed class.

Every particular change to a model element will have a very specific impact, ranging from zero ripple effect, i.e., an entirely local change, to a change affecting every element in the element's impact scope. In the worst case, every element in the entire model could be impacted, if a respective change occurs to a pivotal element, such as a global suptyping root.

As a result, the ECI only attempts to predict average impact values caused by changes to elements c_i, i.e., an average of many actual impact values that can be observed over a long time:

$$\lim_{n\to\infty}\left(\frac{1}{n}\sum_{i=0}^{n} ECI(c_i)\right) = \lim_{n\to\infty}\left(\frac{1}{n}\sum_{i=0}^{n} actual\text{-}impact(c_i)\right) \quad (1)$$

To this end, we use the concept of an *expected value* from probability theory, i.e., the probability-weighted average of individual events x_i:

$$E[X] = \sum_{i=0}^{n} x_i * p_i \quad (2)$$

An elegant way to define ECI according to this notion is to express it as the vector dot product between a probability (likelihood of change) vector and an impact (impact of change) vector:

$$ECI[I] = \vec{L}\cdot\vec{I} = \begin{pmatrix} LoC(c_1)\\ LoC(c_2)\\ \vdots\\ LoC(c_n)\end{pmatrix}\cdot\begin{pmatrix} IoC(c_1)\\ IoC(c_2)\\ \vdots\\ IoC(c_n)\end{pmatrix} \quad (3)$$

5.1.1 Likelihood of Change. The likelihood with which a particular element will be subject to an initial change – which in turn may create ripple effects – depends on the context. Here, we are assuming an open world scenario in which a descriptive model is intended to represent a domain and in which, from time to time, new kinds of individuals are discovered. Each such found individual needs to be accommodated by some type of descriptive model. We could require modelers to assign a probability p_t to each type t in the model, which reflects the likelihood with which they expect t to require a change. This would give us:

$$LoC(t) = p_t$$

However, such an approach would be far too demanding on modelers. Instead, we suggest that modelers only assign a probability p_h to each topmost ancestor root in the system which reflects the probability of a change occurring anywhere in the hierarchy (relative to other hierarchies). This allows modelers to express different levels of stability for respective hierarchies while drastically reducing the effort through computing the probability of the vast majority of elements based on the root probabilities.

For the respective probability computation we are making two assumptions:

(1) a type is less likely to change, the higher up a hierarchy it is positioned.
(2) the probability of siblings in a type's extension or subtype partition is uniformly distributed.

Regardless of whether the classification hierarchy or the subtype hierarchy is concerned, types at higher levels/layers are less likely to require change since their higher level of abstraction provides them with more immunity against the need for revision. From (2) it follows that the combined probability of any of n siblings to require change is independent of n, but that the likelihood for each individual sibling to require change is reduced by the factor $\frac{1}{n}$.

Therefore, assuming that the factor with which the probability of change changes when going upwards the hierarchy is $\frac{1}{f_c}$, i.e., in the case of a single descendant $f_c = \frac{p_t}{p_{ancestor(t)}}$, the probability of an element t changing can be calculated recursively:

$$p(t) = p(t, \frac{p_h(t)}{p_{total}(root(t))})$$

where

$$p(t, p_r) = \begin{cases} p_r\,, & ansc(t) = 0\\ \frac{f_c}{|siblings(t)|} * p(ancestor(t), p_r), & \text{otherwise}\end{cases}$$

and

$$p_{total}(t) = p(t, 1) + \sum_{e\in descendants(t)} p_{total}(e)$$

We are using $p_{total}(root(t))$ to normalise the probability for the top element of the hierarchy such that the total probability of all elements in the hierarchy sums up to $p_h(t)$.

In order to obtain the full formula for LoC, we need to use the above function and employ it for the classification ($\mapsto t\_p(t)$) and subtype hierarchies ($\mapsto s\_p(t)$) respectively. We anticipate that these hierarchies do not contribute to the likelihood of change in equal measure which is why we introduce a type weight factor tw

for the classification hierarchy contribution and then normalize the result with the denominator $1 + tw$:

$$LoC(t) = \frac{1}{1+tw}\,(s\_p(t) + tw * t\_p(t)) \tag{4}$$

Ideally, empirical experimentation should be used to obtain the appropriate tuning for tw. In the absence of any empirical input, one may assume $tw = 1$.

5.1.2 Two-Dimensional Fragility. When a model element is altered during the lifetime of the model, the change might have certain effects on associated elements in its impact scope. In order to illustrate the impact scope of SeniorActor in our example (cf. Fig. 2) the triangles denote the impact scope in terms of the classification hierarchy and the circles denote for the impact scope in terms of the inheritance hierarchy, i.e., SeniorActor's impact scope for the subtyping dimension only contains Actor, but its impact scope in the classification dimension contains BobBrown, EricEchols, AnnSmith, and DanDacosta, plus ActorType .

In order to predict the average "impact of change" (*IoC*) for a given element t, we first define a context-free impact:

$$IoC(t) = i_t$$

The value i_t represents the impact (i.e., effort proxy) of dealing with the change required for element t locally, without involving any other elements.

If the element t has a direct ancestor then there is an average probability with which it can cause a ripple effect on the latter. We define the additional impact caused by such a direct neighbor-induced impact as $i_t * ip\_up$. In other words, ip_up is an impact dampening factor ($ip\_up < 1$). With ip_up set to 0.25, for instance, the effort-of-change proxy for a direct ancestor of t would be 25% of the effort of changing t itself. We expect this impact value to decrease exponentially with the distance of the ancestor. Hence an ancestor with distance d will make the following contribution to the overall impact:

$$Impact\_Contribution(d) = i_t * ip\_up^d$$

The total sum of all ancestor impact factors for an element t with $ansc(t)$ ancestors is therefore

$$IoC_a(t) = \sum_{i=1}^{ansc(t)} ip\_up^i$$

We assume the absence of multiple classification and multiple inheritance, which makes the above formula complete.

Analogously, the total sum of all descendant impact factors for an element t with a single linear descendant chain of depth $desc(t)$ is:

$$IoC_d(t) = \sum_{i=1}^{desc(t)} ip\_down^i$$

Since elements can have multiple direct descendants – whether these are instances or subtypes – we need to sum over all of these descendants and apply IoC_d recursively:

$$IoC_d(t) = \begin{cases} 1, & descendants(t) = \varnothing \\ \displaystyle\sum_{e \in descendants(t)} ip\_down * IoC_d(e), & \text{otherwise} \end{cases}$$

Combining the downward and upward impact contributions in both classification and subtyping dimensions, and multiplying the impact factor contributions with the local impact value i_t yields:

$$IoC = i_t * \begin{pmatrix} 1 \\ + \\ s\_IoC_d(t) + s\_IoC_a(t) \\ + \\ t\_IoC_d(t) + t\_IoC_a(t) \end{pmatrix} \tag{5}$$

There is no need for an explicit weighting factor for the classification dimension in this formula because different contribution strengths can be chosen via respective s_ip_up, s_ip_down, t_ip_up, and t_ip_down factors.

Also, note that we do not normalize the impact with respect to the size of the impact scope since the overall impact actually increases with the size of the impact scope.

5.2 Control Capacity

The second concrete metric we propose is intended to measure the *control capacity* (*CC*) of a multi-level model. Note that independently of the number of levels, bottom level (O_0-level) elements can never control any other elements and top level elements are not (ontologically) controlled by any other elements. Hence, two-level approaches imply *type level (O_1) anarchy* since there are no limitations (outside generic language well-formedness rules) that constrain modeler activity at the (top level) type level. Such a lack of control makes it impossible to ensure any intended structure/organization at the type level. This, in turn, increases error-proneness since it is then possible to inadvertently violate design requirements or fail to adhere to mandatory structural constraints.

Hence it is desirable to use levels O_2 and above to control O_1 level content. The tighter the control exerted by levels higher up the classification hierarchy, the fewer errors can be made.

A very crude approximation for *CC* would be the ratio of clabjects that are ontologically typed versus the total count of clabjects. With C_t as the set of ontologically typed clabjects –

$$C_t = \{c \mid \exists\, T : c \triangleleft T\}$$

– and C the set of all clabjects we have as a first approximation:

$$CC = \frac{|C_t|}{|C|} \tag{6}$$

Note that in a two-level model, no O_1-level clabjects can make a positive contribution to *CC* in terms of being classified by a type. While it is unavoidable to have some number of untyped elements in a non-self-terminating ontological classification hierarchy (i.e., $\frac{|C_t|}{|C|}$ can never reach 100%), a multi-level modeling hierarchy is nevertheless intrinsically better equipped to maximize *CC*.

A more refined approach to calculating *CC* needs to

(1) consider control through supertypes, and
(2) account for the depth of control.

Supertypes impose the presence of features just as (deep) types do. While it could be argued that any element is more likely to have an ontological type than it is to have one or more supertypes, it is most certainly the case that once a supertype is nominated, the

latter exerts some level of control, e.g. guaranteeing the presence of fields, operations, and relationships to other elements.

For both the classification and subtyping dimensions it can be argued that the further the reach of control, the more powerful it can be considered to be. For instance, a deep O_3-level type with potency-3 features will impose more order on the whole multi-level model than a regular O_1-level type. However, we should ensure to not "overcount" control capacity in order to avoid giving too much weight to elements with deep control. If we simply replaced each element that counts as 1 in $|C_t|$ with its ancestor count then the overall sum would entail cumulative contributions, e.g., the deep classification of PythonCoding in Fig. 2 would not just result in a weight of 2 (with TaskType being two levels up), but in 2 + 1 = 3 as the contribution 1 from Coding would also be counted.

Therefore, to calculate CC – for now only taking the classification dimension into account – we only consider elements without descendants:

$$C_\perp = \{c \mid descendants(c) = \varnothing\}$$

We can then define:

$$CC = \frac{1}{|C|} \sum_{e \in C_\perp} ansc(e) \tag{7}$$

Note that Eq. 7 produces the same result as the earlier crude approximation $\frac{|C_t|}{|C|}$ if there are only two levels. Eq. 7 produces more adequate results than the crude approximation if there is a high proportion of deep characterization (since summing up all branch lengths in a tree yields a higher value than counting its edges).

For the full definition of CC we now only need to consider the suptype dimension as well:

$$CC = \frac{1}{|C| * (1 + tcw)} \left(\begin{array}{c} \sum_{e \in C_s \perp} ansc\_s(e) \\ + \\ tcw * \sum_{e \in C_t \perp} ansc\_t(e) \end{array} \right) \tag{8}$$

The weight tcw is used to adjust the contribution from classification to control capacity versus the contribution from generalization. Again, we are not claiming to be in possession of an empirically justified value for tcw. A value of 1 would achieve equal contribution from both dimensions.

6 RELATED WORK

Atkinson and Kühne applied a minimalistic metric – number of elements in a model – when comparing the complexity of multi-level models with two-level models [7]. The underlying assumption was that requiring fewer model elements to describe the same subject indicates less accidental complexity and thus aids understandibility and maintainability.

De Lara et al. included references/associations, stereotypes, and pattern occurrences in their respective size metric [21].

Rossini et al. [31] compared a two-level version of a cloud modeling language called CloudML, which employs the type-object pattern [24], to a multi-level modeling version. The authors also assume that a small model size is desirable because a larger model is harder to comprehend. Of the high-level evaluation criteria they considered, "extensibility" is the one most related to our ECI metric. Rossini et al.'s "extensibility" relates to minimizing impact on related modeling elements when adjusting a model, e.g., to increase its adequacy. They found the multi-level model to be more extensible than its two-level counterpart.

Gerbig compared a multi-level model to a standard two-level UML model [15]. As well as numerous standard object-oriented design metrics [10, 22, 23, 30], he also applied the aforementioned minimalistic accidental complexity metric, applying it separately to different kinds of model elements, e.g., classes versus connections. Gerbig also used the following compound metric definition to measure "complexity": $\frac{AWF+AAP}{DSC}$, where AWF is the average well-formedness rules count, AAP is the average additional operations count, and DSC is the design size (class count). Further high-level model qualities he considered include reusability, understandability, and extendability. For these, Gerbig used the relatively simple compound metrics defined in [23, Tab. 8] and applied them to both a multi-level model and a two-level model.

McQuillan and Power observed that the majority of UML metrics are low-level "counting metrics" [27]. They furthermore point out that some popular metrics such as Halstead's metrics are often criticized for not having been demonstrated to correlate well with high-level model quality attributes. McQuillan and Power moreover point out that some standard software metrics intended to measure code quality can straightforwardly be used on models, mentioning in particular Chidamber and Kemerer [10] which were used by Gerbig (see above).

De Lara et al. proposed and used three quality metrics for level contents – reusability, domain-specificity, and simplicity – when evaluating the impact of model refactorings on these quality aspects [20]. The notion of domain-specificity is related to our notion of control capacity since the former is defined by the relative count of linguistic extensions in a level, i.e., the ratio of elements which are not ontologically classified. De Lara et al. thus maintain that less control over level content indicates increased domain-specificity of said content.

Athena is a language-independent and customizable software metrics tool [33]. Focused on programming languages, Athena supports Pascal, Ada, and C, and not only offers the possibility of measuring a set of kernel metrics but also features a specially designed specification language that allows further design and code metrics to be easily added. We believe that an analogous tool supporting deep model metrics would be very desirable.

Vanderfeesten et. al. presented an approach for applying metrics to business process models [34]. They employed the ProM tool to compute the metrics "complexity", "cohesion", "coupling", and "modularity". In order to evaluate the "cohesion" and "coupling" metrics the ProM tool transforms models into graphs from which the metrics are computed.

The "Maintainability Index (MI)" is a compound metric that was designed to correlate with the maintainability of a system by calculating the weighted sum of four standard metrics [12]. Despite its adoption in Visual Studio, the metric has a number of detractors that doubt its adequacy. Nevertheless, this compound metric is relevant in our context since it attempts to measure a high-level quality, and due to the empirical approach which was used to determine the weight values. A regression analysis was

used to determine the weights based on the values of the lower-level metrics and developer verdicts about the maintainability of the measured systems. We propose that values for deep metric parameters, such as the ones used in Sect. 5, could be obtained in a similar way.

7 FUTURE WORK

In this paper we have outlined some of the issues involved in defining meaningful deep metrics on multi-level models, and have presented examples of what such metrics could look like. This line of work could be continued in a number of directions, including –

- investigating further deep metrics for model qualities,
- expanding the scope to *model adequacy* metrics,
- the consideration of (deep) connections [13], and/or operation bodies,
- developing a universal, abstract representation of multiple multi-level models, that would allow standardized metric definitions to be applied to multi-level models regardless of the native language they are expressed in,
- building an environment using this universal representation approach which is able to accept custom metric definitions, and
- using multiple multi-level models and open world scenarios in order to ascertain empirically motivated values for the "dampening" and "likelihood" parameters used in Sect. 5.

8 CONCLUSION

In this paper we have motivated the utility of *deep metrics* as a way to extend the community's ability to compare multi-level modeling approaches, not only to each other but also against two-level approaches. Beyond supporting such comparisons, deep metrics will also allow comparative evaluations of various models expressed in the same multi-level modeling language, but employing different modeling styles. Even single fine-grained modeler decisions could be supported by quantitative predictions regarding high-level model qualities.

As samples of the kind of metrics we are envisioning, we presented two deep metrics for computing the *expected impact change* and the *control capacity* for models respectively. The novel notion of *control capacity* is particularly destined to be suitable to demonstrate measurable advantages of multi-level models over two-level models. We do not claim that these proposals represent ready-to-employ solutions; certainly we did not attempt to suggest any concrete values for the parameters with which these metrics can be tuned. Instead, our main purpose was to illustrate the value a deep classification dimension can provide and to give an idea of the (very manageable) complexity involved in defining metrics that attempt to correlate with high-level model qualities.

While significant further work is still necessary to make the vision of a community-accessible tool for comparing multi-level approaches a reality, once this kind technology has become available, it will provide an objective method to quantitatively evaluate multi-level models with respect to relevant high-level qualities such as understandability, maintainability, and control capacity.

ACKNOWLEDGMENTS

We thank Colin Atkinson for extensive discussions related to the topics of this paper and comments on an earlier version of this paper. We also thank the anonymous reviewers for their helpful remarks.

REFERENCES

[1] João Paulo A. Almeida, Adrian Rutle, Manuel Wimmer, and Thomas Kühne. The MULTI Process Challenge. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (2019-09). 164–167. DOI: https://doi.org/10.1109/MODELS-C.2019.00027

[2] Colin Atkinson and Ralph Gerbig. 2016. A Feature-based Comparison of Melanee and MetaDepth. In *Proceedings of the 3rd International Workshop on Multi-Level Modelling co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2016) (Multi 2016)*.

[3] Colin Atkinson, Ralph Gerbig, and Bastian Kennel. 2012. On-the-Fly Emendation of Multi-level Models. In *Modelling Foundations and Applications - 8th European Conference, ECMFA 2012, Kongens Lyngby, Denmark, July 2-5, 2012. Proceedings (Lecture Notes in Computer Science)*, Antonio Vallecillo, Juha-Pekka Tolvanen, Ekkart Kindler, Harald Störrle, and Dimitrios S. Kolovos (Eds.), Vol. 7349. Springer, 194–209. DOI: https://doi.org/10.1007/978-3-642-31491-9_16

[4] Colin Atkinson, Ralph Gerbig, and Thomas Kühne. 2014. Comparing multi-level modeling approaches. In *Proceedings of the 1st International Workshop on Multi-Level Modelling co-located with the 17th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2014) (CEUR Workshop Proceedings)*, Vol. Vol-1286. 43–52.

[5] Colin Atkinson and Thomas Kühne. 2001. The Essence of Multilevel Metamodeling. In *Proceedings of the 4th International Conference on the UML 2000, Toronto, Canada (LNCS 2185)*, Martin Gogolla and Cris Kobryn (Eds.). Springer Verlag, 19–33. DOI: https://doi.org/10.1007/3-540-45441-1_3

[6] Colin Atkinson and Thomas Kühne. 2003. Rearchitecting the UML Infrastructure. *ACM Transactions on Modeling and Computer Simulation* 12, 4 (Oct. 2003), 290–321.

[7] Colin Atkinson and Thomas Kühne. 2008. Reducing accidental complexity in domain models. *Software & Systems Modeling* 7, 3 (2008), 345–359.

[8] Colin Atkinson and Thomas Kühne. 2017. On Evaluating Multi-Level Modeling. In *Proceedings of the 4th International Workshop on Multi-Level Modelling co-located with the 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2017) (CEUR Workshop Proceedings, ISSN 1613-0073)*, Vol. Vol-2019. 274–277.

[9] Freddy Brasileiro, João Paulo A. Almeida, Victorio A. Carvalho, and Giancarlo Guizzardi. 2016. Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata. In *Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion)*. International World Wide Web Conferences Steering Committee, 975–980. DOI: https://doi.org/10.1145/2872518.2891117

[10] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering* 20, 6 (1994), 476–493.

[11] Tony Clark, Ulrich Frank, and Manuel Wimmer. 2017. Preface to the 4th International Workshop on Multi-Level Modelling. In *Proceedings of the 4th International Workshop on Multi-Level Modelling co-located with the 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2017) (CEUR Workshop Proceedings)*, Vol. Vol-2019. 210–212.

[12] D. Coleman, D. Ash, B. Lowther, and P. Oman. 1994. Using metrics to evaluate software system maintainability. *Computer* 27, 8 (1994), 44–49.

[13] Thomas Kühne Colin Atkinson, Ralph Gerbig. 2015. A unifying approach to connections for multi-level modeling. In *Proceedings of MODELS'15*. IEEE Computer Society, 216–225.

[14] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. 1994. *Design Patterns: Elements of Object-Oriented Software Architecture*. Addison-Wesley.

[15] Ralph Gerbig. 2017. *Deep, seamless, multi-format, multi-notation definition and use of domain-specific languages*. Ph.D. Dissertation. Universität Mannheim. https://madoc.bib.uni-mannheim.de/42010/

[16] Brian Henderson-Sellers. 1995. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc.

[17] Sallie Henry and Dennis Kafura. 1981. Software structure metrics based on information flow. *IEEE transactions on Software Engineering* 5 (1981), 510–518.

[18] Muzaffar Igamberdiev, Georg Grossmann, and Markus Stumptner. 2016. A Feature-based Categorization of Multi-Level Modeling Approaches and Tools. In *Proceedings of the 3rd Workshop on Multi-Level Modelling co-located with the 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2016) (CEUR Workshop Proceedings)*, Vol. Vol-1722. 45–55.

[19] Thomas Kühne. 2006. Matters of (Meta-) Modeling. *Software and System Modeling* 5, 4 (2006), 369–385. DOI: https://doi.org/10.1007/s10270-006-0017-9

[20] Juan De Lara and Esther Guerra. 2018. Refactoring Multi-Level Models. *ACM Trans. Softw. Eng. Methodol.* 27, 4, Article 17 (Nov. 2018), 56 pages. DOI: https://doi.org/10.1145/3280985

[21] Juan De Lara, Esther Guerra, and Jesús Sánchez Cuadrado. 2014. When and how to use multilevel modelling. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 2 (2014), 1–46.

[22] Mark Lorenz and Jeff Kidd. 1994. *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc.

[23] Haohai Ma, Weizhong Shao, Lu Zhang, Zhiyi Ma, and Yanbing Jiang. 2004. Applying OO metrics to assess UML meta-models. In *International Conference on the Unified Modeling Language*. Springer, 12–26.

[24] Robert C Martin, Dirk Riehle, and Frank Buschmann. 1997. *Pattern languages of program design 3*. Addison-Wesley Longman Publishing Co., Inc.

[25] Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering* 4 (1976), 308–320.

[26] Steve McConnell. 2004. *Code Complete, Second Edition*. Microsoft Press, USA.

[27] Jacqueline A McQuillan and James F Power. 2006. On the application of software metrics to UML models. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 217–226.

[28] Bertrand Meyer. 1997. *Object-Oriented Software Construction (2nd Ed.)*. Prentice-Hall, Inc., USA.

[29] Leonid Mikhajlov and Emil Sekerinski. 1997. The fragile base class problem and its impact on component systems. In *European Conference on Object-Oriented Programming*. Springer, 353–358.

[30] Sandeep Purao and Vijay Vaishnavi. 2003. Product metrics for object-oriented systems. *ACM Computing Surveys (CSUR)* 35, 2 (2003), 191–221.

[31] Alessandro Rossini, Juan De Lara, Esther Guerra, and Nikolay Nikolov. 2015. A comparison of two-level and multi-level modelling for cloud-based applications. In *European Conference on Modelling Foundations and Applications*. Springer, 18–32.

[32] Ian Sommerville. 2018. *Software Engineering* (10. ed.). Hallbergmoos.

[33] C. Tsalidis, D. Christodoulakis, and D. Maritsas. 1992. Athena: A software measurement and metrics environment. *Journal of Software Maintenance: Research and Practice* 4, 2 (1992), 61–81. DOI: https://doi.org/10.1002/smr.4360040202 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.4360040202

[34] Irene Vanderfeesten, Jorge Cardoso, Jan Mendling, Hajo A Reijers, and Wil MP van der Aalst. 2007. Quality metrics for business process models. *BPM and Workflow handbook* 144 (2007), 179–190.