On the Measurement of Design-Time Adaptability for Process-Based Systems

Jörg Lenhard, Matthias Geiger, and Guido Wirtz
Distributed Systems Group
University of Bamberg
Bamberg, Germany
{*joerg.lenhard, matthias.geiger, guido.wirtz*}@*uni-bamberg.de*

***Abstract*—Today, process languages are frequently used for implementing service-oriented systems and a variety of specifications for this task exist. These specifications strive for the portability of processes among different runtime environments, i.e., process engines. However, direct portability, especially of executable processes, is seldom achieved. If processes cannot be ported directly among engines, an option is to adapt them. Such an adaptation is nontrivial and hence automated support is desirable. A first step in this direction is the quantification of the design-time adaptability of a process. This quantification is the goal of this paper. We formally define software metrics for measuring the design-time adaptability of processes and validate them theoretically with respect to measurement theory and construct validity using two validation frameworks. Moreover, we implement the metrics computation for *Business Process Model and Notation* (BPMN) processes and demonstrate their practical applicability with an evaluation of a large set of open source processes.**

***Keywords*-Adaptability; Metrics; BPMN**

I. Introduction

The implementation of service-oriented systems with the help of process-aware information systems [1] is increasingly accepted in practice, not only when it comes to the modeling of business processes, but also in terms of the direct execution of process definitions on dedicated process engines [2]. In a service-oriented context, processes-aware technologies are often used for implementing higher-level and value-added services on the basis of other services. This is achieved by defining the control- and data-flow structure between the invocations of services in a process-based manner, i.e., by their *orchestration* [3]. Such process-based *service orchestrations* form an integral part of a service-oriented architecture [4].

Several competing languages and specifications for building executable process-based systems exist, such as BPMN 2.0.2 [5] or the *Web Services Business Process Execution Language 2.0* (BPEL) [6]. BPMN has recently been accepted as an ISO standard and is facing rising adoption in practice. In a BPMN process, dedicated tasks for service invocation or message sending and reception exist, thereby enabling the construction of service orchestrations. Tasks in BPMN are not strictly tied to a specific service implementation technology, as is the case for BPEL and Web Services. This allows for a higher technological flexibility when implementing process-based and service-oriented systems.

A commonality of the different process specifications is that multiple implementations supporting process execution are available and evolve with varying speed. In such a situation, a lot of benefits for the users of an engine in terms of execution performance, license cost, or scalability can be gained by using the most advanced and suitable runtime engine available. To leverage these benefits, existing processes have to be ported from one engine to another. Despite the fact that portability is a central goal of process standards [7], this task is often quite complicated. Frequently, direct porting is not possible, even if a process fully complies to a standard, due to differences in the implementations of that standard. This can happen despite the fact that the implementations claim to support the same standard. For instance, high deviations in standard conformance have been demonstrated for implementations of BPEL [8]. As a consequence, instead of direct porting, processes have to be manually adapted at design-time to enable their execution on another engine.

Our intention is to support the task of adapting processes through techniques of software measurement. We propose several software quality metrics that can be used to quantify the *degree of adaptability* inherent to a process. This provides benefits, such as:

1) Metrics can be used to inform developers about issues in, and the quality of, the processes they implement. When integrated into the development through techniques such as *continuous inspection* [9], developers become aware of problems in the process code. This awareness can increase the quality of the code in the long term. Put into the context of this paper, we try to inform developers about the adaptability of the processes they build.
2) Metrics are a basis for comparing different processes and the prerequisite for quality ranking. They can be used as mechanism for decision support, for instance if it is worth to invest time in adapting a given process.

In [10], we put forward a first idea for evaluating the design-time adaptability of processes, in particular processes written in BPMN. We proposed to capture an adaptability score for every language element and to aggregate this score to an adaptability metric. Here, we build upon this proposal by fully implementing the approach from [10] and testing several mechanisms of metrics computation, complemented with a thorough and comprehensive validation and evaluation. The research question we are trying to answer is:

What kind of metric performs best for quantifying the design-time adaptability of process-based systems?

We provide a theoretical validation of proposed metrics with two validation frameworks and conduct a large size experiment with real-world processes that allows us to separate useful metrics from less useful ones. That way, we can answer the research question.

The next section discusses work related to this paper, followed by the formal definition of our proposed metrics. Thereafter, we theoretically validate the metrics in Section IV and practically in Section V, including the final decision on which metrics are most appropriate. Finally, Section VI concludes the paper with a summary and remarks on future work.

II. Related Work

Work related to our approach separates in two areas which we discuss in the following sections: On the one hand, this is work on specifications and languages for modeling and implementing processes. On the other hand, it is work on software quality models in general and on the measurement of adaptability and metrics for process-based and service-oriented systems in particular.

A. Process Languages and Specifications

Major international standards for implementing process-based systems are the *XML Process Definition Language 2.2* (XPDL) [11], and, as mentioned in the introduction, the *Web Services Business Process Execution Language 2.0* [6] and the *Business Process Model and Notation 2.0.2* [5].

XPDL [11] is promoted by the *Workflow Management Coalition* and addresses the storage and interchange of processes among different tools, especially editors and modeling environments. Runtimes for this language do exist, but are not very common and the language is mainly relevant to the modeling of processes.

BPEL [6] is a language built for the Web Services ecosystem and has received a lot of attention since its conception. Its primary purpose is the task of service orchestration and today a large variety of BPEL runtimes do exist. Being tailored to Web Services technology, the importance of this language is declining in recent years. Owing to its wide practical adoption, it is, nevertheless, a good example of the dichotomy of a specification and its implementations. In previous work [8], [12], we could show that the standard conformance of BPEL runtimes and consequently the portability of BPEL processes varies strongly.

Implementations of BPMN [5] are increasingly adopted today and more and more vendors try to implement the specification. The language was initially intended for the visualization of processes to ease the communication among different stakeholders, but has broadened its scope with the addition of different types of processes, such as process choreographies and also executable processes. For these reasons, we assess the approach in this paper with BPMN, but emphasize that the metrics are independent of it and, hence, applicable to any process language in general.

B. Software Quality, Adaptability, and Metrics for Process-Based Systems

Adaptability has long been recognized as a quality characteristic of software and is part of many software quality models, e.g., [13]–[15]. It is often related to the quality characteristic of *portability*. In this paper, we build on the *SQuaRE* method [13], the new ISO/IEC 25010 series of standards for software quality evaluation. This series revises the well-known ISO/IEC 9126 [16] standard for software quality and is intended as its replacement. The revision is currently underway and although the quality model [13] is already published, standards that describe concrete software metrics [17] are not yet publicly available. The quality model defines *adaptability* as a subcharacteristic of *portability*, next to characteristics such as *replaceability* or *installability*. Our long-term goal is to allow for the quantification of each of these characteristics for process-based systems and we have dealt with portability [18] and installability [19] in prior work.

In [13, p.15], adaptability is defined as the "*degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments*". Our focus here are adaptions to the software environment (i.e., to the code of a process to enable its execution on a different platform). In other words, we focus on *design-time* changes to a process. This opposes other common definitions of adaptability. For example, in adapter synthesis [20], adaptability refers to whether an adapter for a given pair of processes or services can be created. In autonomous systems [21], adaptability refers to whether a system can change its structure to cope with changing requirements, such as a different system load, at runtime.

Adaptability metrics defined in ISO/IEC 9126 [16] focus predominantly on the observation of user behavior, for instance, to see if users can adapt easily to a new software environment. Here, we try to determine how easily a process written in a specification language can be changed, in a way that still complies to the specification, to enable it to run on a different implementation of the specification. Related studies that measure design-time adaptability [22], [23] try to do so by providing an adaptability score at the level of atomic system elements and to aggregate this score to a global degree of adaptability. This is also the path we take here, although our focus is not the adaptability of software architectures as in [22], [23], but of process-based systems.

Lastly, change patterns [24] and the ADEPT project [25] aim to improve the flexibility and robustness of processes. Both primarily address runtime changes of process instances, whereas we concentrate on design-time changes, there denoted as process schema evolution. Change patterns on the one hand define editor operations that should be available for adapting processes and on the other hand specify constructs that make the runtime adaption of processes easier. Here, we do not define such operations or structures, but try to quantify the design-time adaptability of a process.

III. On the Measurement of Adaptability

A common conceptual approach for measuring adaptability explained in [10] and used in similar studies [22], [23], is to start at the level of an atomic system element, for instance by tagging every element with an *adaptability score*. Then, these atomic scores are aggregated at one or more levels to indices or *degrees of adaptability*, until the complete system is considered as a whole. This approach is outlined in Fig. 1.

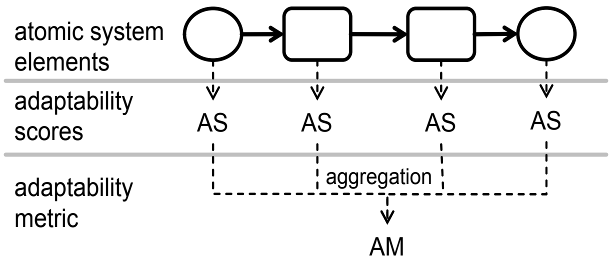

Figure 1. Mechanism for Computing Adaptability

The focus here is on process-based systems, hence, we need to consider the adaptability of atomic process elements and to aggregate their adaptability to a global value for a complete process. The atomic elements of a process are its activities and similar language constructs, such as gateways or events [26]. So, when evaluating the adaptability of processes written in a particular language, it is first necessary to assign an adaptability score to each type of element in that language. The idea we put forward in [10] and expand upon here is to consider the *amount of alternative representations* a process language offers for a certain element. The more of such alternative representations exist, the easier and more likely it is that one can find a semantically equivalent alternative when modifying the process for porting. In other words, the more alternatives exist, the more adaptable the element is. A practical example for BPMN are *receive tasks*, atomic tasks that consume a message. One straight-forward alternative for such a task, among others, is an *intermediate message catch event* which provides semantically identical behavior.

We define the adaptability score $AS(e)$ as follows:

Definition 1 (Adaptability Score): $AS(e)$ is the cardinality of the set of alternatives $\{alt_1^e, \ldots, alt_n^e\}$ available for an element e.

$$AS(e) = \mid \{alt_1^e, \ldots, alt_n^e\} \mid, where \qquad (1)$$

- e is an element (an activity, event, or gateway) of the process language.
- alt^e is a semantically equivalent alternative to language element e.

This score needs to be determined for every language element. The next step towards computing an adaptability metric as depicted in Fig.1 is the normalization and aggregation of the scores for all elements of a complete process.

A. Definition of the Metrics

For the definition of an adaptability metric based on adaptability scores, two crucial problems are to be solved:

1) *How to turn the scores into a meaningful relative value?* Absolute values are common for software metrics, but often hard to interpret. This is much easier with relative values, which range in a certain interval, as for instance percentage values. That way, values close to the upper bound of the interval can be identified as referring to high quality and vice versa.
2) *How to normalize this value with respect to process size?* Different real-world processes can be expected to have different sizes. Metric values should be normalized with respect to process size, because otherwise they cannot meaningfully be used to compare processes of different sizes.

The first problem can be solved by introducing an *adaptability degree* $AD(e)$ that turns an adaptability score as defined in the previous section into a relative value:

Definition 2 (Adaptability Degree): The adaptability degree AD of an element e is a function that maps an element e to a value in the interval of zero and one, i.e., a percentage scale.

$$AD(e) \rightarrow [0, 1] \qquad (2)$$

It is a design choice of our approach to map degree values to the interval of $[0, 1]$. Strictly speaking, we could choose any interval, but the reason we choose $[0, 1]$ is one of understandability. This interval resembles a percentage scale and this scale is easily understood by most people. This eases the interpretation of the metric to some degree and thus lowers the barrier for its adoption. The question of what values in this scale refer to *high* or *low* depends on how the mapping is achieved. This is the central difference between the different metrics we introduce below. Hence, we redefine the adaptability degree for each adaptability metric in the following sections.

The second problem can be solved by following the approach taken by previous studies [22], [23] and computing the arithmetic mean of all adaptability degrees of the different elements of a process. Hence, an adaptability metric $AM(p)$ of a process p is defined as follows:

Definition 3 (Adaptability Metric):

$$AM(p) = \begin{cases} 0 & \text{if } p = \emptyset \\ \overline{AD(e_1),\ldots,AD(e_n)} & otherwise \end{cases} \tag{3}$$

- p is a process that consists of the elements $e_1,\ldots,e_n$, i.e., it corresponds to the set of elements, which make up the process $p = \{e_1,\ldots,e_n\}$
- For an empty process, i.e., $p = \emptyset$, $AM(p)$ is defined as zero, i.e., $AM(p) = 0$
- For an nonempty process, i.e. $p = \{e_1,\ldots,e_n\}$, $AM(p)$ is defined as the arithmetic mean of adaptability degrees of the elements of the process: $\overline{AD(e_1),\ldots,AD(e_n)}$

In the following sections, we propose different ways of computing an adaptability degree for a given process element, and hence different adaptability metrics.

B. Binary Adaptability Degree

The first way for computing adaptability degrees is a *binary mapping*. Here, $AD(e) \rightarrow \{0,1\}$ applies, i.e., every element is mapped to a degree of zero or one. Such a mapping is, for instance, used in [22].

This mapping is achieved by applying a threshold on the adaptability score for an element. The score, as specified in Def. 1, is an absolute representation of the adaptability of an element, the higher the better. When comparing the elements of one language, it is possible to distinguish high adaptability scores with respect to all scores of the language from low ones. In the binary mapping, we assign a degree value of one to elements with high scores and a value of zero to elements with low scores. To achieve this, a suitable threshold value needs to be found. For instance, the threshold can be defined at 50%. This means that elements that have lower adaptability scores than 50% of all language elements are mapped to zero and elements that have higher or equal scores to 50% of all language elements are mapped to one. Put formally:

Definition 4 (Binary Adaptability Degree):

$$AD_{binary}(e) = \begin{cases} 1, & \text{if } AS(e) \geq (t * R) \\ 0, & otherwise \end{cases}, where \tag{4}$$

- $R \in \mathbb{N}_0$; R is a reference value, the maximum adaptability score achieved by any element in the language under consideration, i.e., $\forall e, AS(e) \leq R$
- $t \in]0,1[$; t marks a threshold that is used to discriminate between high and low adaptability values

As a consequence, a *binary adaptability metric* $AM_{binary}(p)$ is based on the binary adaptability degree: $AM_{binary}(p) = \overline{AD_{binary}(e_1),\ldots,AD_{binary}(e_n)}$. An appropriate threshold value t that results in meaningful metric values can only be fixed based on practical experiments. In Section V, we evaluate several different thresholds and select the most appropriate one.

C. Weighted Adaptability Degree

The binary degree maps the score values to the boundaries of the interval of $[0,1]$. The purpose of a *weighted mapping* is to utilize the full scale of this interval. That way, the resulting metric might provide a more fine-grained and precise quantification of adaptability. We define the weighted aggregation as follows:

Definition 5 (Weighted Adaptability Degree):

$$AD_{weighted}(e) = AS(e)/R, where \tag{5}$$

- R refers to the maximum adaptability score as defined in Def. 4
- $\forall e, AS(e) \leq R$; implies that $AD_{weighted}(e) \rightarrow [0,1]$

As a result, the adaptability degree of every language element is relative to the most adaptable element of the language. This leads to the desired normalization to the interval of $[0,1]$. A *weighted adaptability metric* $AM_{weighted}(p) = \overline{AD_{weighted}(e_1),\ldots,AD_{weighted}(e_n)}$ is based on the weighted adaptability degree.

IV. Theoretical Validation

Validation of software metrics is a crucial task [27]. A theoretical validation clarifies the properties of the metrics and is the basis for the selection of the proper statistical methods for an interpretation. Here, we use two theoretical validation frameworks for validating our metrics [28], [29]. Both of these frameworks are frequently used in studies similar to this one, e.g. [18], [19], [30], and address *measurement theory* and *construct validity*.

A. Measurement Theory

The first validation framework [28] specifies different categories of code metrics. Its aim is to "*make the measure definition process more rigorous and less exploratory*" [28, p. 71]. It summarizes the work of previous approaches that define desirable metric properties and has been conceived primarily for object-oriented systems. Nevertheless, it is applicable for process-based systems as well and has been used for this purpose in similar studies, e.g. [18], [19]. The framework takes the generic view of a system S that consists of a set of elements E and a set of relations R among them, i.e., $S = < E, R >$, $R \subseteq E \times E$. This metaphor can directly be applied to process-based systems, where a system S corresponds to a process, the set of elements E to the set of activities, gateways, and events of the process, and the set of relations R among elements to the control-flow relations of the process graph.

The framework defines several metric categories, being *size*, *length*, *complexity*, *cohesion*, and *coupling*. Furthermore, it lists the formal properties that metrics of each category should fulfill. The metrics we define here fit best into the category of complexity metrics, although we take a slightly different view on the nature of complexity. According to [28], the complexity of a system originates from the relationships among system elements only, whereas atomic elements have no inherent complexity. In our point of view, complexity originates from the relationships among system elements *and* their inherent complexity (in terms of this paper, their inherent adaptability) in combination. Notwithstanding, the framework is still applicable here. A second difference of our metrics to the complexity metrics of [28] results from the fact that we normalize our metrics with respect to the size of the system. For [28], complexity metrics yield absolute values. The purpose of the normalization here is to enable the direct comparison of systems of different size, as discussed in Section III-A. Hence, we deal with *normalized complexity metrics* similar to the metrics presented in [18].

According to [28], complexity metrics should fulfill the properties of *nonnegativity*, *null value*, *symmetry*, *monotonicity*, and *additivity*.

1) Nonnegativity: Complexity metrics should yield nonnegative values. Since the adaptability score is defined as the cardinality of a set, it is always nonnegative. Moreover, adaptability degrees are mapped to the interval of $[0,1]$. The arithmetic mean of a set of values in this interval, i.e., an adaptability metric as defined in Def. 3, is always nonnegative.

2) Null value: The complexity of an empty system should be zero. In our case, an empty system corresponds to a process p with no elements, i.e., $p = \emptyset$. For this case, $AM(p)$ is defined as zero in Def. 3.

3) Symmetry: The labeling for representing the relationships among system elements should not affect the metric value. Here, the labeling refers to the ordering of elements in the process graph. The reordering of elements in the process graph, without altering control-flow semantics, is possible for a variety of elements, for instance in the case of parallelism. Symmetry means that two processes $p =< E,R >$ and $p' =< E,R' >$ with identical elements E and control-flow semantics but different element ordering R and R' should have the same metric value. We compute adaptability on a per-element basis through the adaptability degree. This degree is fixed independently of an elements position in the process graph, so symmetry holds: $AM(p) = AM(\{e_1,\ldots,e_n\}) = AM(p')$

4) Monotonicity and Additivity: When two unrelated parts of a system are taken together, the resulting complexity should at least be equal to the sum of the combined parts. This means that complexity metrics should be additive and monotonous. Additivity holds for adaptability degrees, but not for the metrics, due to normalization. We apply the arithmetic mean to achieve a normalization with respect to the size of a process and adding up the mean values of two process fragments is not meaningful. Hence, additivity in the sense of [28] does not hold for reasons of normalization. Nevertheless, adaptability metrics are still monotonous. For instance, let $p^1 = \{e_1,\ldots,e_i\}$ and $p^2 = \{e_j,\ldots,e_n\}$ be two unrelated parts of process p:

$$\begin{aligned} AM(p) = AM(p^1 \cup p^2) = AM(\{e_1,\ldots,e_n\}) = \\ \frac{AD(e_1,) + \ldots + AD(e_i) + AD(e_j) + \ldots + AD(e_n)}{N_{|p^1 \cup p^2|}} \\ \geq min(AM(p^1), AM(p^2)) \end{aligned}$$

The metric value of two process fragments taken together cannot be lower than the metric value of the less adaptable of the two fragments and therefore the metrics are monotonous. This differs slightly from the definition in [28] which requires that metric values should be at least as high as the sum of the values of the fragments, but does not conflict with monotonicity, as they still are always higher than the lower of the two values.

To summarize the above discussion, adaptability metrics are normalized complexity metrics which fulfill the properties of nonnegativity, null value, symmetry, and monotonicity.

B. Construct Validity

The second validation framework [29] addresses *construct validity*. It helps to clarify whether the metrics actually measure what they are intended to measure. The framework takes a qualitative approach and specifies ten questions that have to be answered for a proposed metric. By *attribute*, [29] refers to the quality characteristic to be measured and by *measurement instrument* the authors refer to the tool with which metric values are computed. In the following, we answer the questions for our metrics:

1) What is the purpose of the metrics? The purpose is the facilitation of private self-assessment and improvement, and the information of developers and system administrators about the adaptability characteristics of a process. When having to port a process, the metrics can help to make the decision whether it is worth to invest in its adaptation.

2) What is the scope of the metrics? The scope of the metrics is a single project of one workgroup. The metrics are applicable during and after development.

3) What attribute are the metrics trying to measure? The metrics try to measure the *adaptability* of a process, the ease with which the elements of a given process can be modified at design-time without changing the execution semantics of the process.

4) What is the natural scale we are trying to measure? The scale of the attribute is intrinsic to the attribute itself and independent of the way in which we try to quantify it. At the current time, we cannot tell what the scale of adaptability is, but it can reasonably be assumed that processes differ in their adaptability in a way that allows us to construct a natural ordering. Hence, the attribute of adaptability at least has an ordinal scale.

5) What is the natural variability of the attribute? We do not know in which ranges adaptability typically varies. Being a technical attribute inherent to a process, we know that it is not subject to variation common for human attributes, such as developer productivity depending on the time of the day.

6) What are the metrics and measurement instrument? All the metrics, the functions that assign values to the attribute, are formally defined in Section III-A. Adaptability scores of elements are fixed values and obtained through *counting*. Adaptability degrees and metrics are computed based on this. The *measurement instrument* is a static analyzer in which we implemented the metrics computation, explained in the practical evaluation in Section V.

7) What is the natural scale of the metrics? The metrics are defined in Def. 3 on an interval scale of $[0,1]$.

8) What is the natural variability of the instrument? This question refers to the *measurement error* of the instrument, in our case a static analyzer. Since we compute the metrics through static analysis, there is no variability of the instrument for subsequent analyses of the same process. There is no way in which we can guarantee the absence of errors in our software. For instance, programming errors that impact the metric values the tool produces might exist. We try to limit the amount of errors by open sourcing the tool and making all of the code available to public scrutiny, and by providing an excessive set of unit tests for the tool itself. Another source of measurement error are the adaptability scores encoded in the tool. These are based on human judgement and, naturally, we can err in our understanding of the semantics of BPMN elements. We tried to minimize these errors through in-group peer review.

9) What is the relationship of attribute and metric value? Our metrics are *directly* [27] related to process elements. Given elements with a higher degree of adaptability are introduced into the process, this will be detected by our measurement instrument and the resulting metric value will increase accordingly. As discussed in Section III, the adaptability of an element is related to the number of alternatives available to that element.

10) What are side effects of the measurement instrument? Humans are good at modifying their behavior when being measured to produce more desirable metric values without changing the underlying attribute. Since we measure process code, no such influence is possible.

V. Practical Evaluation

The purpose of the practical evaluation is to see how the different metrics perform in practice and is an important part of their validation [27]. It allows to accomplish a meaningful selection of useful metrics and thresholds and allows to see which practical properties hold for the different metrics. In the next section, we first describe the design and instrumentation of the evaluation, followed by the discussion of the results.

A. Design and Instrumentation

Experiments in software engineering typically follow a certain scheme [31, pp. 85ff.]. The goal of this evaluation is to analyze real-world processes for the purpose of assessing the metrics and selecting the most appropriate ones. To meet this goal, we evaluate software that is finished and available, i.e., we perform an off-line experiment. Using the evaluation, we can validate several quality factors for the different metrics, addressed by the following hypotheses:

1) *The weighted adaptability computation improves discriminative power in comparison to a binary computation:* The ability to distinguish between different pieces of software, the *discriminative power*, is an important quality factor of a software metric. We expect that the weighted computation, due to more fine-grained scoring, outperforms the binary computation with respect to this quality factor.
2) *The metrics can be used for comparing processes of different size:* Another important quality factor of a software metric is the ability to allow for a comparison of programs of very different size. This is often problematic, especially with complexity metrics [32].
3) *The metric values are resilient to minor changes in the underlying data:* Minor modifications to process code should not result in significant changes to the metric value, since this would imply that the mechanism of computation is unstable.

A practical evaluation needs to be based on real-world process models and, therefore, on a particular process language. We use the BPMN 2.0.2 specification [5] for this evaluation, but any of the languages discussed in the related work section would have been suitable. BPMN currently is the most dynamic in terms of new and evolving runtime environments and for this reason, we base our evaluation on BPMN.

To implement the approach, it is necessary to set adaptability scores, as defined in Def. 1, for every relevant element of BPMN. This language offers a variety of different process diagrams, but since our software metrics are specially directed at executable artifacts, we limit the evaluation to BPMN *processes* [5, pp. 145–314]. We reviewed the specification and set scores for all elements of processes, that is *activities*, including *tasks* and *subProcesses*, *gateways*, and *events*. Thus, we cover all control-flow aspects of BPMN, but abstract from data-flow in this evaluation. Moreover, we omit language elements that are relevant to the visualization of the process only, such as *lanes*. The result is a total of 94 language elements for which we set a score. Thereafter, we extended our metrics suite for computing portability metrics, the *prope* tool[1], with a static analyzer that detects the occurrence of the BPMN elements important here and performs the

[1]The project page of prope is located at http://uniba-dsg.github.io/prope/. This page also contains information on how to execute the metrics suite and analyze BPMN processes.

computation of the defined adaptability metrics. This tool is publicly available and all adaptability scores can be examined.

As the basis for a practical evaluation, we need concrete data in the form of BPMN processes. We gathered this data from numerous open source projects with the help of the Openhub Open Source network[2]. This network currently indexes more than 650 thousand open source projects and allows to search their code. To obtain our primary data set, we queried the network on May 15th 2014 for files that:

1) have the file extension `bpmn`, `bpmn2`, or `xml`,
2) contain the keyword *definitions*, the top-level element of a BPMN-compliant file, and
3) contain the BPMN 2.0 namespace.

We downloaded and analyzed the resulting files using the static analyzer. We tried to parse every file downloaded from Openhub and if we could find a valid *definitions* element and at least one *process* element beneath it, we computed the adaptability metrics for it. In total, this resulted in 2997 BPMN processes. We re-performed the query on October 10th 2014, to obtain the same data set at a later point in time. This is necessary for evaluating hypothesis 3.

In the following sections, we first describe the nature and construct usage of the gathered processes. Thereafter, we present and discuss descriptive metric data and evaluate several hypotheses to determine the most useful metrics.

B. BPMN Processes and Usage of Elements

The processes we obtained can be seen as representative of the open source usage of BPMN, since they are all gathered from freely accessible projects. We performed several correctness checks, such as the correct usage of namespaces and the validation of element references, on the processes and excluded them from the analysis if they did not pass these checks. For instance, despite the fact that a file contains the correct BPMN 2.0 namespace, a process defined in it might use a different one. The amount of such processes in our data set is negligible and more than 99% of the analyzed processes are using the correct BPMN namespace. Moreover, we validated element references in the processes using the *BPMN-Reference-Validator*[3], and could detect issues, such as an *EventDefinition* referenced in an event but not found in the process, in 8% of the processes. This step reduced the initial set of processes from 2995 to 2745. Finally, 12% of the processes, a total of 327, are explicitly marked as executable, i.e., the *isExecutable* flag of the *process* element is set to *true*.

In an influential paper [33], which also had impact on the current version of BPMN [34], zur Muehlen and Recker analyzed the usage of BPMN elements in process models and found that only a very small subset of the existing elements were actually used in practice. As this also has implications on our metrics, we reproduced this analysis for the processes at hand. The plot in Fig. 2 depicts the occurrence frequency of selected BPMN elements, i.e., the percentage of processes in which the elements occur. We limit the figure to the ten most frequent elements found in the processes. Although the evaluation from [33] uses an older

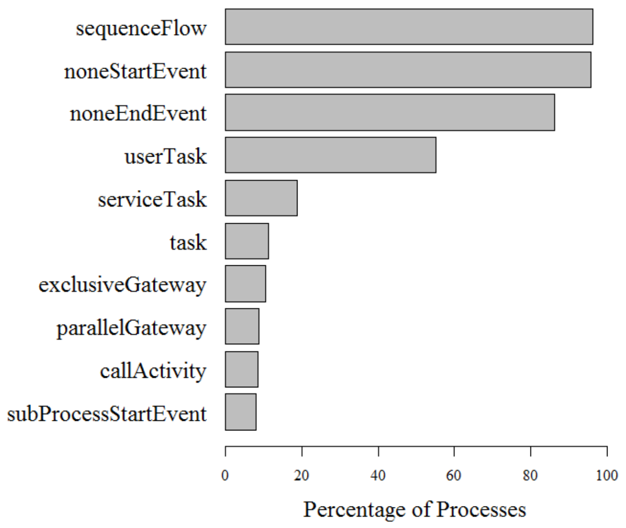

Figure 2. Occurence Frequency of BPMN Elements

revision of BPMN, our results largely reinstate theirs. By far the most common elements in BPMN processes that occur in almost all processes are *sequence flows* and ordinary *start-* and *end events*. The next most frequent elements are two specific types of *tasks*. This is similar to [33] where *tasks* are not separated by a specific type. The occurrence frequency of elements is important to the metrics computation for the following reason: The most frequent elements will need to be present in every process and adapting them is simply not an option. If included in the adaptability computation, these elements would introduce noise into the metric value. For instance, *sequence flows* are very common in any process and including them in the computation would introduce a strong weighting towards the adaptability of *sequence flows* in general. Since they cannot be adapted, this weighting would be noise. Based on the data depicted in Fig. 2, we can exclude the most common elements from the metrics computation, being the elements that occur in more than two thirds of all processes. These are *sequence flows* and ordinary *start-* and *end events*.

C. Comparison of the Metrics

Table I lists descriptive statistics[4] for the processes, sorted into executable and nonexecutable processes, i.e., processes

[2]More information on the network can be found on the network page located at http://www.openhub.net/.

[3]This is a validation tool for BPMN 2.0 files. Please see the project page located at http://www.uni-bamberg.de/pi/bpmn-constraints for more information.

[4]All statistical computations in this paper were performed using the R software [35]. The scripts that produce the data presented here are also publicly available as part of the static analyzer. That way, the results presented here can be reproduced.

Table I
STATISTICS DATA FOR PROCESS LIBRARIES (ROUNDED TO TWO DECIMAL PLACES)

Process Group	N	Statistics	$AM_{bin.}$, $t=0.2$	$AM_{bin.}$, $t=0.4$	$AM_{bin.}$, $t=0.6$	$AM_{bin.}$, $t=0.8$	$AM_{weighted}$
executable	327	Mean	0.91	0.88	0.60	0.09	0.56
		Std. Dev.	0.14	0.16	0.15	0.16	0.14
		Disc. Pow.	0.08	0.09	0.12	0.09	0.21
nonexecutable	2418	Mean	0.96	0.93	0.59	0.09	0.59
		Std. Dev.	0.12	0.14	0.19	0.16	0.12
		Disc. Pow.	0.02	0.02	0.02	0.01	0.05

that have the *isExecutable* flag set to *true* or *false* respectively. We computed the binary metric as defined in Section III-B with the thresholds set at 20%, 40%, 60%, and 80% of the most adaptable element of the language to see which threshold performs best, and the weighted metric as defined in Section III-C. The setting of these thresholds is a design choice and it might be valuable to test their sensitivity at a more granular level in future work. The metric values are very similar for the two process groups, but there are strong differences among the different metrics. A first step is to examine if there are significant differences between the adaptability of executable and nonexecutable processes. To determine this, we first performed the Shapiro-Wilk test to see if the metric values are normally distributed. Since this is clearly not the case[5], we chose a nonparametric test, the Mann-Whitney U test [36] to find out if there are differences in the distributions of the metric values for the two process groups. The null hypothesis here is that there are no significant differences in the distributions of the metric values for the two process groups. Here, p-values do not reach a significant level for any of the metrics, so the null hypothesis cannot be rejected and there seem to be no significant differences between executable and nonexecutable processes in the data at hand. The reason for this might be that most processes we obtained are in fact used for execution, but not marked with the respective flag. In the following, we evaluate the previously stated hypotheses regarding the quality of the metrics.

1) Hypothesis 1 – Discriminative Power: We can show that our metrics perform better when analyzing processes marked for execution and we can filter for the metrics that are particularly good. This can be achieved by looking at the *discriminative power* of the metrics, their ability to differentiate between different processes. The better a metric differentiates between different processes, the better it can be used for quality comparison and ranking of processes. To determine the discriminative power, we removed all duplicate metric values from the different data sets, resulting in a list of unique metric values for each metric and process group. By comparing the amount of unique values to the total amount of values, we obtained the percentage of unique values for the processes in the different sets. These percentage values are listed in the third row of each process group in table I. It can be seen that the discriminative power is much higher for executable processes and in particular the weighted adaptability degree performs best. Looking at the binary degrees, a threshold set at 60% yields best results. This narrows the scope of the useful metrics to the weighted degree and the binary degree with a threshold set at 60%.

[5]Due to space limitations, we omit the presentation of the results of the Shapiro-Wilk normality test here.

Table II
DESCRIPTIVE STATISTICS AND MANN-WHITNEY U TEST FOR DIFFERENCES IN PROCESS SIZE

Process Group	N_{small}	N_{large}	Statistics	$AM_{bin.}$, $t:60\%$	$AM_{weighted}$
exec.	62	80	$\tilde{x}_{small}$	0.67	0.62
			$\tilde{x}_{large}$	0.58	0.53
			p	0.004	0.09
			U	2559	2803
nonexec.	573	564	$\tilde{x}_{small}$	0.67	0.62
			$\tilde{x}_{large}$	0.43	0.54
			p	$2.2e^{-16}$	$2.2e^{-16}$
			U	220285	213544

2) Hypothesis 2 – Process Size: A further quality characteristic of software metrics is the ability to compare systems of different size. Typically, metrics tend to produce very different values when the system size differs a lot [32], thus rendering them insufficient for comparing systems of a very different size. To investigate how well our metrics perform in the face of processes of different size, we extracted two sets of processes. These are small and large processes separated into executable and nonexecutable groups. The set of small processes refers to the first quartile with respect to the number of elements in the process, i.e., the 25% smallest processes. In the same fashion, the set of large processes corresponds to the fourth quartile with respect to the elements in the process. We use the Mann-Whitney U test [36] as above to see if the metrics for small and large processes differ in their distribution. The null hypothesis here is that there are no significant differences in the distributions of the metric values for the two process groups. We limit this comparison to $AM_{weighted}$ and AM_{binary} with a threshold set at 60%.

Table II depicts the results of this test separated by process groups. Because we are doing four tests, we have to adjust the significance level to $0.05/4 = 0.0125$. In all but one case,

Table III
DESCRIPTIVE STATISTICS AND ACCURACY FOR REPEATED MEASUREMENTS

Process Group	N_1	N_2	Statistics	$AM_{bin.}$, $t: 60\%$	$AM_{weighted}$
			$\tilde{x}_1$	0.63	0.59
exec.	327	565	$\tilde{x}_2$	0.63	0.57
			Acc.	0	0.04
			$\tilde{x}_1$	0.67	0.62
nonexec.	2418	1921	$\tilde{x}_2$	0.67	0.62
			Acc.	0	0

p-values become significant, meaning that the distributions of the process groups for the metrics really are different. Consequently, these metrics should be treated with care when comparing processes of very different size. The only exception to this is $AM_{weighted}$ for executable processes. Here, no significant differences of the metric values for small and large processes could be detected. This means that the metric really can be used for comparing processes of different size.

3) Hypothesis 3 – Stability: A third quality factor for the metrics addresses their resilience to variances over time. Repeated measurements should produce similar results to demonstrate the stability of the mechanism of computation, i.e., the *predictability* of the metric value. [27, p. 12] recommends to evaluate this aspect by checking if metric values of repeated measurements differ only up to a certain *accuracy* threshold. This can be evaluated with the following formula:

$$\left| \frac{AM(p_2) - AM(p_1)}{AM(p_2)} \right| < Acc.Threshold \qquad (6)$$

By comparing the metric values of different snapshots of our data set over time, we can evaluate this aspect. For this reason, we repeated the data gathering described in Section V-A five months later and obtained a second snapshot of the data. This data set is slightly smaller, with 2719 instead of 2997 processes, but contains a larger amount of executable processes with 565 instead of 327 processes. Since we do not compare single processes, but different sets of processes, we replace the metric values in equation 6 with the median values of the different sets. As [27] leaves no hint for a suitable accuracy threshold, we set it to 0.05, meaning that the difference in metric values of repeated measurement should be no higher. The median values and results are depicted in table III. In nearly all cases, median values are identical. In case of the weighted metric for executable processes, the result is 0.04 which is still below the threshold.

Summarizing the previous paragraphs, we can frame an answer to the research question posed in the introduction. The weighted adaptability metric $AM_{weighted}$ performs best with respect to discriminative power, allows for the comparison of processes of different size, and produces stable results. Hence, it is the metric we recommend for quantifying the adaptability of executable processes.

VI. SUMMARY AND CONCLUSION

In this paper, we proposed and formally defined a set of metrics for measuring the design-time adaptability of process-based systems. We validated the proposed metrics theoretically with respect to measurement theory and construct validity. We implemented the metrics computation for BPMN processes and gathered a large set of real-world processes for a practical evaluation. Based on this evaluation, we could select one metric which performed best.

Various directions of future work follow from this. First, threshold values of the binary adaptability metric could be tested at a more granular level to determine the sensitivity of the threshold value. Second, adaptability is just one part of a measurement framework for evaluating the portability of processes. The construction of this framework is our long term goal. Other quality characteristics, such as replaceability, still remain open for quantification. Third, the testing of the metrics computation for process languages other than BPMN would also be desirable. Fourth, enabling effort prediction based on the metrics would be valuable. Most effort prediction models estimate effort based on lines-of-code. Since we quantify adaptability based on atomic process elements, which correspond to a higher-level notion of lines-of-code, it can be expected that there is a relation between effort and our metrics.

REFERENCES

[1] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley, 2005.

[2] W. M. P. van der Aalst, "Business Process Management: A Comprehensive Survey," *ISRN Software Engineering*, pp. 1–37, 2013.

[3] C. Peltz, "Web Services Orchestration and Choreography," *IEEE Computer*, vol. 36, no. 10, pp. 46–52, October 2003.

[4] M. P. Papazoglou and D. Georgakopoulos, "Service-oriented Computing," *Communications of the ACM*, vol. 46, no. 10, pp. 24–28, October 2003.

[5] ISO/IEC, *ISO/IEC 19510:2013 – Information technology - Object Management Group Business Process Model and Notation*, November 2013, v2.0.2.

[6] OASIS, *Web Services Business Process Execution Language*, April 2007, v2.0.

[7] R. Khalaf, A. Keller, and F. Leymann, "Business processes for Web Services: Principles and applications," *IBM Systems Journal*, vol. 45, no. 2, pp. 425–446, 2006.

[8] S. Harrer, J. Lenhard, and G. Wirtz, "Open Source versus Proprietary Software in Service-Orientation: The Case of BPEL Engines," in *International Conference on Service Oriented Computing*. Berlin, Germany: Springer Berlin Heidelberg, 2013.

[9] P. Merson, A. Aguiar, E. Guerra, and J. Yoder, "Continuous Inspection: A Pattern for Keeping your Code Healthy and Aligned to the Architecture," in *3rd Asian Conference on Pattern Languages of Programs*, Tokyo, Japan, March 2014.

[10] J. Lenhard, "Towards Quantifying the Adaptability of Executable BPMN Processes," in *Proceedings of the 6th Central-European Workshop on Services and their Composition*, Potsdam, Germany, February 2014.

[11] WfMC, *Process Definition Interface – XML Process Definition Language*, August 2012, v2.2.

[12] S. Harrer, J. Lenhard, and G. Wirtz, "BPEL Conformance in Open Source Engines," in *IEEE SOCA International Conference on Service-Oriented Computing and Applications*. Taipei, Taiwan: IEEE, December 17-19 2012.

[13] ISO/IEC, *Systems and software engineering – System and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, 2011, 25010:2011.

[14] T. Gilb, *Principles of Software Engineering Management*. Addison Wesley, 1988, ISBN-13: 978-0201192469.

[15] J. McCall, P. Richards, and G. Walters, "Factors in Software Quality – Concept and Definitions of Software Quality," General Electric Company, Sunnyvale, California, USA, Tech. Rep., 1977.

[16] ISO/IEC, *Software engineering – Product quality – Part 1: Quality model*, 2001, 9126-1:2001.

[17] ——, *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality*, 2013, 25023.

[18] J. Lenhard and G. Wirtz, "Measuring the Portability of Service-Oriented Processes," in *17th IEEE International Enterprise Distributed Object Computing Conference (EDOC2013)*, Vancouver, Canada, September 2013.

[19] J. Lenhard, S. Harrer, and G. Wirtz, "Measuring the Installability of Service Orchestrations Using the SQuaRE Method," in *IEEE International Conference on Service-Oriented Computing and Applications*. Kauai, Hawaii, USA: IEEE, December 16-18 2013.

[20] Z. Zhou, S. Bhiri, H. Zhuge, and M. Hauswirth, "Assessing Service Protocol Adaptability Based on Protocol Reduction and Graph Search," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 9, pp. 880–904, 2011.

[21] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challanges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, 2009.

[22] N. Subramanian and L. Chung, "Metrics for Software Adaptability," in *Software Quality Management Conference*, Loughborough, UK, April 2001.

[23] D. Perez-Palacin, R. Mirandola, and J. Merseguer, "On the Relationships between QoS and Software Adaptability at the Architectural Level," *Journal of Systems and Software*, vol. 87, no. 1, pp. 1–17, 2014.

[24] B. Weber, S. Rinderle, and M. Reichert, "Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems," *Data and Knowledge Engineering, Elsevier*, vol. 66, no. 3, pp. 438–466, July 2008.

[25] P. Dadam and M. Reichert, "The ADEPT project: a decade of research and development for robust and flexible process support," *Computer Science – Research and Development*, vol. 23, no. 2, pp. 81–97, 2009.

[26] M. Weske, *Business Process Management: Concepts, Languages, Architectures (Second Edition)*. Springer-Verlag, Berlin, Heidelberg, 2012, ISBN: 978-3642286155.

[27] IEEE, *IEEE Std 1061-1998 (R2009), IEEE Standard for a Software Quality Metrics Methodology*, 1998, revision of IEEE Std 1061-1992.

[28] L. Briand, S. Morasca, and V. Basily, "Property-based software engineering measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68–86, 1996.

[29] C. Kaner and W. Bond, "Software Engineering Metrics: What Do They Measure and How Do We Know?" in *10th International Software Metrics Symposium*, Chicago, USA, September 2004.

[30] D. Basci and S. Misra, "Measuring and Evaluating a Design Complexity Metric for XML Schema Documents," *Journal of Information Science and Engineering*, vol. 25, no. 5, pp. 1405–1425, 2009.

[31] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer, Berlin, Heidelberg, 2012, ISBN: 978-3642290435.

[32] K. E. Emam, S. Benlarbi, N. Goel, and S. Rai, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Transactions on Software Engineering*, vol. 27, no. 7, pp. 630–650, 2001.

[33] M. zur Muehlen and J. Recker, "How Much Language is Enough? Theoretical and Practical Use of the Business Process Modeling Notation," in *Advanced Information Systems Engineering (CAiSE)*, Montpellier, France, June 2008.

[34] ——, "We Still Don't Know How Much BPMN is Enough, But We Are Getting Closer," in *Seminal Contributions to Informations Systems Engineering: 25 Years of CAiSE*, 2013, pp. 445–451.

[35] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org

[36] H. B. Mann and D. R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," *Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.