An Analytical Framework for Evaluating Service-Oriented Software Development Methodologies

Mikhail Perepletchikov, Caspar Ryan, and Zahir Tari

School of Computer Science and IT, RMIT University, Melbourne, Australia
Email: {mikhail.perepletchikov, caspar.ryan, zahir.tari}@rmit.edu.au

***Abstract*—Service-Oriented Computing is becoming a paradigm of choice for implementing enterprise-level distributed applications, with a number of methodologies having been proposed to provide systematic guidance for the development of service-oriented solutions. However, presently, there is a lack of well-defined and pragmatic Service-Oriented (SO)-specific methodology evaluation approaches, making it difficult to evaluate and compare exiting methodologies in an informed manner. To this end, this paper proposes an analytical framework for evaluating and comparing SO development methodologies using a set of qualitative features and quantitative ratio-scale metrics. A case-study was conducted to demonstrate the practical application of the framework.**

***Index Terms*—Service-Oriented Computing (SOC), Software development methodologies, evaluation framework.**

I. Introduction

Service-Oriented Architecture (SOA) [8] or Service-Oriented Computing (SOC) [44] is a development paradigm for implementing distributed and enterprise class systems, which employ software services as independent and reusable building blocks that collectively represent a software application. SOC is founded on the idea of discovery and composition whereby an executable business process can discover at runtime the most suitable services and orchestrate them in order to satisfy a particular domain or business requirement [34]. This flexibility can lead to new economic opportunities as software functionality is increasingly commoditised as Software as a Service (SaaS) in the Cloud [32].

Developing SOA-based systems, which are typically heterogeneous, large, and complex, is a challenging and time-consuming exercise [9]. Add to this the ever-increasing emphasis on non-functional requirements, as well as the need to support newer delivery models (such as SaaS in the Cloud), and Service-Oriented (SO) system development is only going to become more difficult [1]. To this end, *development methodologies* can bring formal and clearly specified processes that can instruct all or part of the software engineering life cycle. Specifically, incorporating a well-defined and mature methodology into the software development process can result in two important benefits: *i) increase of developer productivity*; and *ii) product quality improvements* [10]. Also, development methodologies provide support for building software systems aligned with specific paradigms and associated logical and technological constraints, which is especially important for newer and yet to be thoroughly understood and documented paradigms such as SOC [46].

A number of methodological approaches have been proposed and applied when developing SO solutions for different business domains, e.g. [1, 8-9, 43]. However, existing approaches are relatively new and typically lack theoretical foundation and empirical evaluation, making it difficult to establish the strengths and weaknesses of the incorporated activities, work products, and other methodological principles. This is further exacerbated by a *lack of SO-specific methodology evaluation mechanisms*. Specifically, the presently available evaluation methodology approaches, e.g. [7, 13, 26], are not sufficient to capture the unique requirements and constraints of SOC [46]. Also, such approaches suffer from a number of *limitations* (e.g. lack of objectivity) as discussed further in Section 2.2.

Therefore, this paper proposes a novel analytical framework that supports the objective evaluation and comparison of software development methodologies using *feature analysis* [24, 27] and *metrics-based assessment* [14, 51]. Specifically, the proposed framework prescribes a set of desirable features that allow *qualitative* examination of various methodology characteristics (e.g. maturity and usability), and a set of *quantitative* ratio-scale metrics for assessing the internal properties of a methodology such as the structural dependencies between its tasks and work products.

To support a more structured and focused evaluation process, the included features and metrics were organised in terms of a *hierarchical quality model* that supports methodology evaluation from three distinct perspectives, "*structured artefact*", "*underlying process*", and "*deliverable product*" as discussed further in Section III. Moreover, the framework was designed to be *generic* (i.e. it includes quality characteristics applicable to *any* software development methodology), and at the same time *customisable* insofar as those characteristics that are paradigm-specific can be adapted and applied to methodologies covering different development paradigms as has been done in this paper for the specific case of SOC. Additionally, the proposed framework is *flexible,* since it can be employed as part of different evaluation strategies, such as quantitative experiments or qualitative

doi:10.4304/jsw.8.7.1642-1659

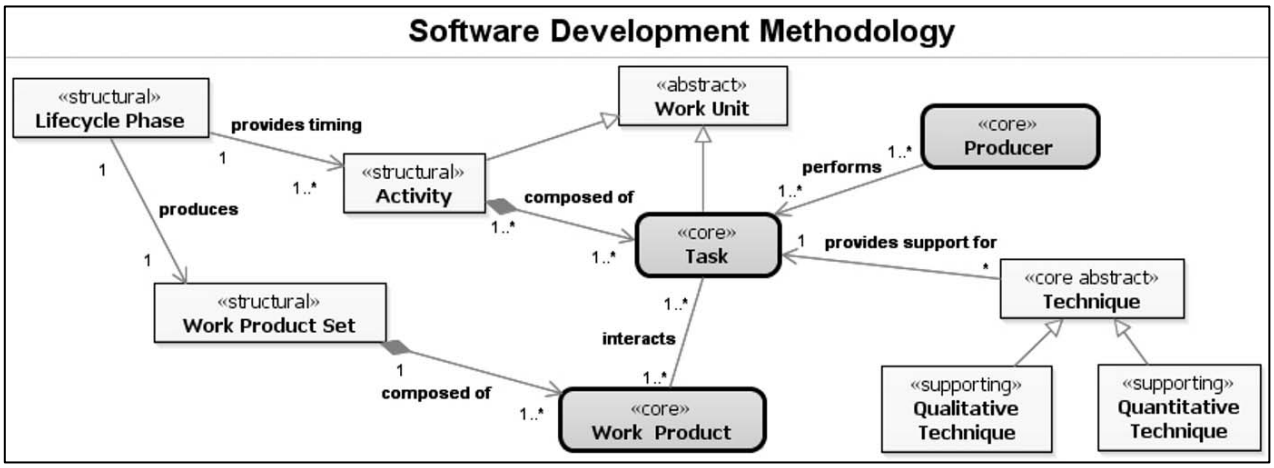

Fig. 1. A meta-model of the software development methodology

case studies and surveys [27], each requiring varying level of assessment expertise and effort, and producing different levels of evaluation accuracy. Finally, for this paper, an exploratory *case study* (see Section VII) was employed as a proof of concept to demonstrate the practicality of the framework by using it to evaluate a subset of a mature and widely-used SO development methodology, SOMA [1].

This paper is organised as follows: Section II defines a generic meta-model of a software development methodology, and reviews existing evaluation approaches including the discussion of their strengths and weaknesses. Section III provides an overview of the proposed framework, followed by the internal details of three covered evaluation perspectives, *structured artefact*, *underlying process*, and *deliverable product,* presented in Sections IV-VI respectively. The practical application of the framework, via an exploratory case study, is presented in Section VII; while Section VIII closes with a summary conclusions and discussion of future work.

II. Background

2.1 A Meta-model of a Development Methodology

A meta-model of a software development methodology was derived in order to *provide a theoretical foundation and consistent formalism for the definition of methodology evaluation mechanisms* in Sections IV-VI. This meta-model, shown in Fig. 1, combines core concepts and definitions extracted from existing mature methodological approaches (such as the Rational Unified Process (RUP) [29] and Object-Oriented Process, Environment and Notation (OPEN) [12]) into one generic model comprising a collection of process *classes* that represent significant elements of a development methodology, the *relationships* between them, and the applicable *constraints* (e.g. a Task must interact with at least one Work Product).

The proposed meta-model is documented using UML 2 notation and takes an OO approach (i.e. it uses standard class relationships, such as association and generalisation). This is in line with the "Software and Systems Process Engineering Meta-Model 2.0 (SPEM)" [41] specification that prescribes a comprehensive process engineering meta-model, defined as a set of generic UML 2 stereotypes, for modelling and enacting software development methodologies and their components. However, in contrast to SPEM, which includes more than fifty different process classes and has a relatively complex hierarchical structure, the meta-model presented in this section is simple and intuitive covering only the main process classes necessary to support the proposed analytical framework.

The process classes themselves are categorised into 4 different types (indicated as UML stereotypes in Fig. 1):

(1) *Core* classes that represent fundamental measurable components of any development process.

(2) *Structural* classes that provide logical structure for the relevant core classes, thereby supporting more targeted methodology evaluation insofar as the evaluation procedure can be applied to the individual structures of interest (e.g. lifecycle phases).

(3) *Abstract* classes that serve as conceptual placeholders for a set of related process classes.

(4) *Supporting* classes that provide assistance to the core classes.

Specifically, according to the proposed meta-model, a software development *methodology* (**M**) incorporates a number of *lifecycle phases* (**LP**) that represent major logical partitions of the development process and provide a natural organisation and timing to the execution of development activities (**A**), and the production of work product sets (**WPS**).

The work product sets are composed of collections of related work product (**WP**) types that represent anything of value that is *produced, modified, or reviewed* during the execution of the development process. The work product types can support both *input* and *output* interactions with the development tasks (**T**) that symbolise low-level functionally cohesive executable process operations encapsulated into and time boxed by the corresponding activities.

The tasks are performed by the producers (**P**) which can be either human participants or dedicated software tools that perform automated manipulation of work

products. Finally, the tasks are supported by optional techniques **(Te(T))** which can be classified as *quantitative* (e.g. software metrics or transformation algorithms) or *qualitative* (e.g. set of informal design guidelines).

The detailed definitions of all process classes, together with associated examples, can be found in Appendix A.

2.2 Existing Methodology Evaluation Approaches

A number of approaches (e.g. [7, 13, 26]) have been proposed for evaluating and comparing software development methodologies with the aim of:

(1) Providing an understanding of particular strengths of a methodology.

(2) Assisting methodology enhancement by identifying shortcomings.

(3) Supporting an informed and structured comparison of methodologies.

(4) Allowing extraction of valid process fragments (i.e. WU and WP) from existing methodologies for the purpose of assembling a new methodology.

According to the DESMET [27] meta-methodology for evaluating software engineering methods and tools, methodology evaluation processes can be broadly classified into two categories, referred to as *internal* and *external* in this paper, and described further below.

Internal evaluation is applied to the methodology itself in order to establish its validity or conformance to some accepted norm (e.g. as captured by a set of required features or ontological representations, see Sections 2.2.1 and 2.2.2). Such evaluation allows examining *process, (structured) artefact, and (deliverable) product aspects of methodologies*. That is, although a methodology prescribes a *process* used to build software products, it can also be considered as a constructed *artefact* structured in terms of a collection of inter-dependent meta-classes (see Section 2.1), and delivered as a physical *product* comprised of documentation, supporting tools, and training; as explained further in Section III.

External evaluation is performed to establish the impact, or measurable effects, of methodologies when applied in practice. Such evaluation aims to quantify the impact of a given methodology on: i) the *quality of the produced work product instances* (i.e. $\text{Quality}(WP_{[instance]})$); and ii) the process efficiency as reflected by the productivity of the producers (P) in terms of a *number of produced work product instances per unit of time* (i.e. $\text{Efficiency}(P \rightarrow WP_{[instance]})$).

There are three main methodology evaluation approaches that can be employed as part of internal or external evaluation processes (or both) – *feature analysis, ontological evaluation, and metrics-based assessment*.

2.2.1 Feature Analysis

Feature analysis is a widely referenced *informal and qualitative* methodology evaluation approach [24, 27, 53]. Using this technique, the evaluators extract a set of important features from available methodologies and combine them into an evaluation checklist, which is then applied to methodologies either within the same development paradigm or across paradigms [24]. The evaluation itself is done by examining the structure and documentation of selected methodologies, and assigning an ordinal-scale number or category to a given checklist feature (or evaluation criterion).

The strength of this strategy is that it is *easy and fast to execute* if the set of criteria is well defined [27]. Also, this approach is *flexible* insofar as it can be used as part of internal and external evaluation processes. Moreover, due to its simplicity, the feature analysis technique is commonly used to evaluate methodologies defined for newer and/or not well-understood development paradigms [53]. For example, the Methodology Evaluation Framework for Component-Based Development (CBD) [7], which was proposed at the time when CBD was still in its inception, used feature analysis to evaluate whether a given methodology can be deemed CBD-specific. Also, more recently, [18] proposed a set of evaluation features, distilled from the research literature, that cover some of the unique characteristics and design requirements of Service-Oriented software engineering.

A major limitation of feature analysis is its inherent subjectivity. Firstly, the evaluation checklists are usually *developed based on the subjective opinions of the evaluators* with limited formal justifications provided for the inclusion of particular features, making it difficult to assess their completeness and internal consistency [56]. Secondly, the actual evaluation process (i.e. criteria application) is commonly conducted in an informal manner, thereby heavily depending on how the evaluators *subjectively score the methodology features against the checklist criteria [24]*.

2.2.2 Ontological Evaluation

Ontological evaluation [15, 17, 26] is based on the idea of evaluating the constructs (e.g. work products (WP) and work units (WU)) of existing methodologies by matching them with pre-defined ontological constructs. More specifically, the ontological evaluation can be applied at two different levels of abstractions, *representation* and *design process*, as follows:

(1) Representation level [58] – the expressive power of a given methodology is evaluated for its completeness and clarity in respect to a predefined ontology [58]. *Completeness* in the context of ontological evaluation refers to the ability of a methodology's grammar (i.e. language/s used to specify work units and work products) to describe all the prescribed ontological constructs. *Clarity* refers to the degree to which the methodology's grammar can be interpreted unambiguously. For example, [58] proposed a generic representational model (BWW) capturing fundamental concepts of any information system using 29 ontological constructs (e.g. system, subsystem, properties, event, etc.). They also introduced the concepts of construct overload, redundancy, excess, and deficit that allow formal reasoning about the strength of a mapping between specific methodologies and relevant ontology. Those concepts were then used to evaluate the ontological expressiveness of IS analysis and design grammars. In later work, [17] demonstrated the process of applying this generic ontological model for the purpose of determining the ability of the selected integrated process modelling grammar to provide "good

representations of the perceptions" of business analysts.

(2) Design process level [15] – the expressive power of a given methodology is evaluated for its ability to represent generic design processes, and other applicable concepts of the Function-Behaviour-Structure (FBS) [15] framework that accounts for the situatedness of the design, viewing it as a dynamic activity driven by interactions between designers and the work products being designed. Specifically, FBS subdivides the design process into five distinct sub-processes: formulation, synthesis, analysis, evaluation, documentation, and three reformation types that link functions, behaviours and structures at expected and actual levels.

The main advantage of ontological evaluation is that it provides *stronger theoretical foundation* and *formal evaluation semantics* compared to feature analysis [15]. However, ontological evaluation *requires a formal representation of a methodology grammar* to be available, which is not always the case given that a majority of existing methodologies are documented and managed using natural language descriptions rather than formal specifications [41]. To this end, producing a valid formal grammar for a given methodology can be a *difficult and time consuming* task. Additionally, assuming that the methodology grammar is readily available, the tasks of classifying grammar fragments into the specific ontological constructs and evaluating the strengths of their mapping are still *subjective*.

Finally, the validity of existing ontology models themselves is questionable. For example, it was suggested that BWW ontology [58] is over-engineered and at the same time not capable of capturing the essence of specific paradigms and modelling objectives [17]. To this end, there may be a need to extend and tailor existing ontological models, which can be a *time consuming* exercise especially for the newly emerged paradigms [26], such as SOC, that lack accepted design principles.

2.2.3 Metrics-based Assessment

Metrics can provide a formal and objective mechanism for both *internal* (e.g. [14]) and *external* (e.g. [21, 52]) methodology assessment.

Internal assessment, in this context, involves analysing the *complexity* of software development methodologies (M) and incorporated work products (WP) and tasks (T) using a set of dedicated ratio-scale metrics. However, at present, there is a *lack of theoretically valid and empirically evaluated methodology-specific internal metrics*. To our knowledge, the "Framework for the modelling and evaluation of software processes" (FMESP) [14] is the only available approach that provides a basic suite of validated (using the Distance-based framework [49]) and empirically evaluated metrics for quantifying the *cognitive*[1] *complexity of a given methodology (M)*. Specifically, FMESP treats the structure of a methodology as a bi-directional graph, where vertices symbolise three fundamental process classes, activities (A), work products (WP), and producers (P); and edges correspond to the relationships between these entities (i.e. $P->_{performs}T$ and $T<->_{interacts}WP$, see Section 2.1).

To measure this structure, FMESP proposed a suite of twelve metrics, *six* of which indicated a statistically significant correlation with the perceived cognitive complexity of investigated methodologies: i) *NPR* (number of producers involved in the methodology application); ii) *NA* (number of activities); iii) *NWP* (number of work products); iv) $NDWP_{in}$ (number of input dependencies of the work products with the activities); v) $NDWP_{out}$ (number of output dependencies of the work products with the activities); and vi) *NDWP* ([total] number of work product and activity dependencies).

In addition to FMESP, Rossi and Brinkkemper [51] proposed a suite of ratio-scale metrics for quantifying the *total conceptual complexity of the individual work product (WP) types* modelled in terms of three main meta-types: i) *objects (O)* (e.g. service interface); ii) *properties (P)* (e.g. interface name and interface operations); and iii) *relationships (R)* (e.g. "exposes" and "uses"). Note that although their metrics were defined in a systematic and formal manner [51], they were not theoretically validated or empirically evaluated.

As stated previously, the application of any software development methodology is said to result in two major *external* benefits [10]: i) product quality improvements; and ii) increase of productivity. In previous work, the product quality was shown to be affected by its structural design properties [47-48], and as such, a number of researchers proposed evaluating methodologies in terms of their *impact on the structural properties of the produced work products (WP)*. For example, design metrics have been previously used to compare the structure (e.g. complexity) of OO class diagrams that resulted from applying different OO development methodologies to the same problem domain [52]. Similarly, software metrics (e.g. from ISO/IEC 9126 standard [19]) can be used to measure the productivity of producers (P) in order to determine the *impact of a particular methodology on the overall process efficiency*.

The main advantage of both internal and external metrics-based assessment is that it supports *formal*, *objective*, and *automated* methodology evaluation. However, the number of currently available *internal* metrics is limited, and the derivation and validation of new metrics can be a *significant and complex research undertaking*. Also, *external* metric values can only be calculated by observing the (real-life) behaviour of producers (P) applying the methodology in practice, and/or evaluating the quality of the produced (complete) work products. This type of evaluation is also more *resource intensive* compared to the feature analysis and ontological techniques [27].

[1] Cognitive complexity can be defined in terms of the level of effort needed to understand a given methodological component (e.g. task (T)) and apply it in practice [51].

TABLE 1.
EXISTING METHODOLOGY EVALUATION APPROACHES - SUMMARY

Evaluation Approach	Evaluation Target					Evaluation Process
	Internal <*Artefact*>	Internal <*Process*>	Internal <*Product*>	External <*Product Quality*>	External <*Process Efficiency*>	
Metrics	**<C> * +** (e.g. [14])	**<C> ***	**<C> ***	**<C> +** (e.g. [6, 20])	**<C> +** (e.g. [21])	• High effort • Formal • Objective
Feature Analysis	=	**<C> * +** (e.g. [27])	**<C> * +** (e.g. [27])	=	=	• Low effort • Informal • Subjective
Ontological	*Not Applicable*	**+** (e.g. [15], [30])	**+** (e.g. [17], [58])	*Not Applicable*	*Not Applicable*	• High effort • Formal • Subjective

<C> Covered by the proposed framework; ***** Introduced in this paper; **+** Previously available ; = Can be defined in future work;

2.2.4 Summary and Limitations

The examined evaluation approaches, *feature analysis, ontological evaluation,* and *metrics based assessment*, are summarised in Table 1. Specifically, this table: i) shows which specific approaches are covered by the framework proposed in Sections IV-VI; ii) illustrates overall applicability of a given approach to different internal and external evaluation targets, and iii) indicates the associated degrees of evaluation effort, and the objectivity and formality of the evaluation procedures and ensuing outcomes. For example, *metrics* based assessment can be deemed as both *formal and objective* given that the evaluation procedure is supported by the means of automated and formally defined ratio-scale measures. However, such assessment requires a *high degree of effort* to derive and validate new internal metrics, or obtain data necessary to calculate the values of external metrics. In contrast, although *feature analysis* can be employed with *minimal effort* as part of the initial evaluation process, this approach is inherently *informal and subjective*. Finally, *ontological evaluation* can potentially replace or enhance feature analysis with more *formal* and targeted evaluation semantics. However, the process of producing methodology grammars; and then classifying grammar fragments into ontological constructs is *subjective and resource-consuming*.

The evaluation framework proposed in this paper follows the *feature analysis and metrics-based assessment* approaches, which can *cover a broad spectrum of internal and external evaluation targets* as shown in Table 1. Moreover, these approaches can be considered *complementary* insofar as the evaluation of a methodology can be performed: i) informally with minimal required resources, using the feature analysis approach; and ii) in a formal and structured, but more resource-intensive manner as warranted by the application of metrics. Note however that in future work the framework *could be augmented with ontological constructs* so as to provide a more formal foundation for the evaluation process.

III. METHODOLOGY EVALUATION FRAMEWORK - OVERVIEW

The proposed framework incorporates a set of unambiguous *qualitative features* defined on a five point ordinal Likert scale, and a collection of quantifiable *ratio-scale metrics*. These features and metrics were identified based on a comprehensive and critical analysis of the related literature (see Section II), and then expanded with a set of characteristics extracted from existing SO methodological approaches (e.g. [1-2, 8-9, 43]).

To provide a formal foundation for the framework derivation and inclusion of specific features and metrics, and to support a more focused evaluation process, the decision was made to structure the framework according to existing software measurement [11] and quality [3, 19] models [2] that treat the concept of quality using a *hierarchical structure* where quality is divided into a number of quality characteristics which are then further decomposed into measurable sub-characteristics. Such hierarchical structure can support a more focused evaluation process, insofar as the evaluators can concentrate on the specific evaluation aspects based on the available resources and overall evaluation goals and requirements. This is also in line with the DESMET [27] meta-methodology that suggests that the evaluation mechanisms can be complex concepts in themselves, and thus should be decomposed into conceptually simpler items structured in a hierarchical manner. Moreover, it has been previously suggested that hierarchical models are easier to analyse and maintain [25].

To this end, the framework is structured in terms of a *hierarchical quality model*, shown in Fig. 2, comprising *three quality perspectives* (*structured artefact, underlying process, and deliverable product*) that encapsulate *seven high-level quality characteristics* (C1-C7) subdivided into *twenty-two sub-characteristics* (SC1.1-SC7.4) that can be directly assessed using ordinal features or ratio-

[2] Evaluating a methodology using the feature analysis and metric-based approaches is similar to assessing its quality. This is because in the area of empirical software engineering any evaluation process is said to involve measurements of entities and their quality characteristics [11].

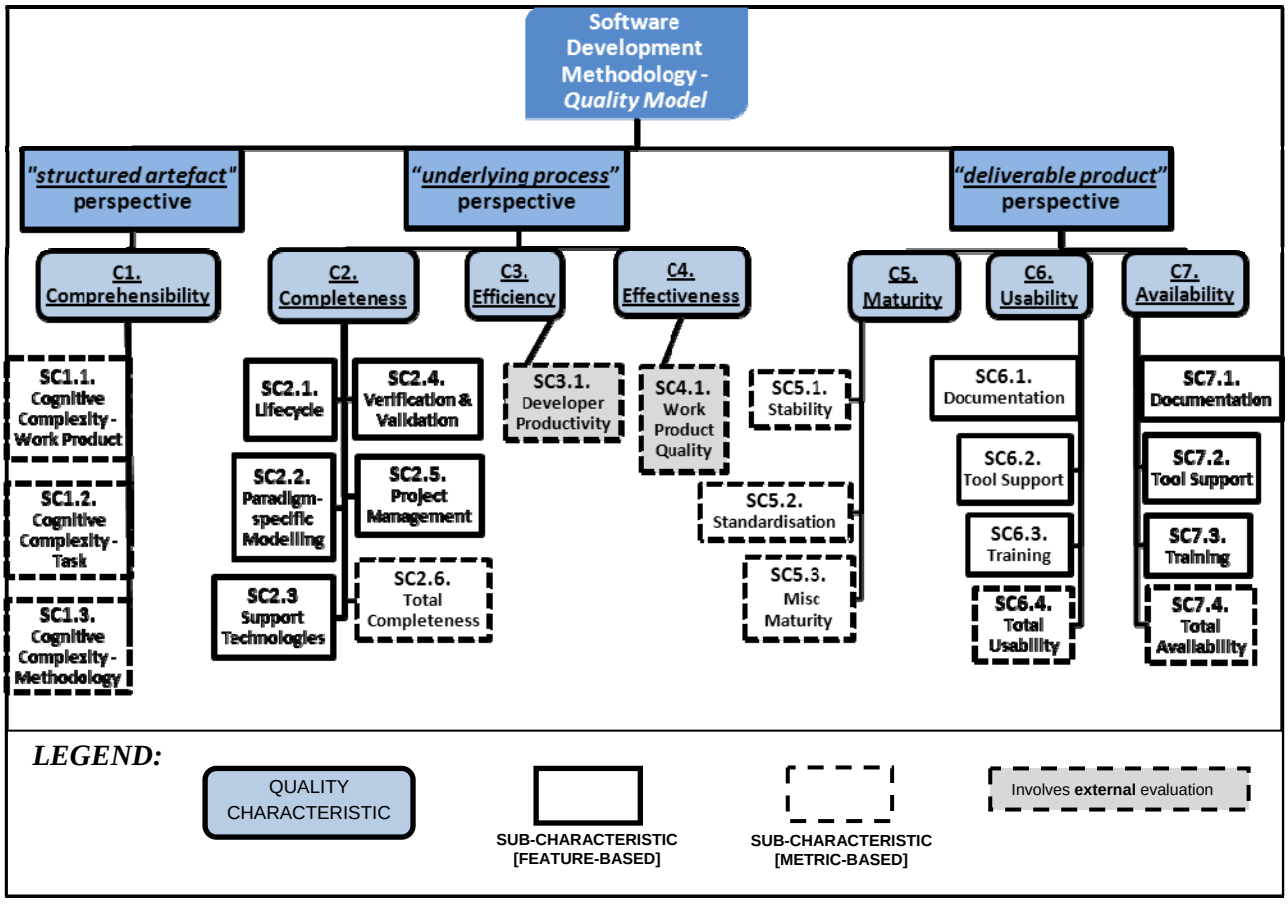

Fig. 2. SO methodology evaluation framework – Quality Model: perspectives, characteristics, and sub-characteristics

scale metrics:

- The "structured artefact" perspective treats a methodology as a constructed entity (or artefact) structured in terms of a collection of process *classes* and their *relationships* as covered by the meta-model of a software development methodology proposed in Section 2.1. Specifically, this perspective aims to evaluate the comprehensibility (Characteristic C1) of *any* generic software development methodology (M) and incorporated tasks (T) and work products (WP) by quantifying their *cognitive complexity.*

- The "underlying process" perspective, on the other hand, covers various characteristics of the prescribed development process and associated methodological guidance, and is designed to be *tailored and adapted for a particular development paradigm* as has been done in this paper for the specific case of SOC. For example, the *"Completeness"* characteristic (C2) incorporates the *"Lifecycle"* sub-characteristic (SC2.1), which evaluates methodologies with respect to their coverage of the core development lifecycle phases of SOC - Service Identification, Service Specification, and Service Realisation [1] (see Appendix B).

- The "deliverable product" perspective treats a methodology as a *physical (deliverable) product* available in the form of three main components - documentation, supporting software tools, and training. The *usability* and *availability* of those components, as well as the *maturity* of the methodology as a whole, are evaluated by the proposed framework (Characteristics C5-C7).

Finally, the proposed evaluation mechanisms (features and metrics), grouped into specific quality sub-characteristics, can be applied at two different levels of granularity: i) individual task (T) and/or work product (WP); and ii) methodology (M) as a whole. The low-level (i.e. T and WP) mechanisms are intended to support *targeted methodology assessment* by identifying specific *strengths and shortcomings of provided methodological components* (e.g. T and WP). In contrast, the high-level (i.e. methodology) mechanisms include aggregated values for the low-level features/metrics, as well as broader evaluation mechanisms that cover generic aspects of methodologies (e.g. maturity, as reflected by sub-characteristics SC5.1-5.3), and are designed to be a useful tool for *comparing different methodologies in an objective and comprehensive manner.*

IV. Methodology Evaluation Framework – "Structured Artefact" Perspective

This section covers the *structured artefact* quality perspective and its sole characteristic (*Characteristic C1. Comprehensibility*), and associated sub-characteristics (denoted by the *SC1.n* marker) and metrics (denoted by *SC1.n-Mn*). The proposed sub-characteristics are described individually below in terms of the rationale behind their

inclusion in the framework, and measurement specifics covering the scale, granularity (WP, T, or M), and definitions of the provided metrics. Note that both scale and granularity will be the same for all the metrics included in a given sub-characteristic. Also, all included metrics are of the internal type (refer to Section 2.2).

4.1 Characteristic C1. ***Comprehensibility***

In this framework, comprehensibility is defined as the cognitive effort of users to understand the internals of the individual methodological components (i.e. WP and T) and analyse the overall structure of the prescribed process (e.g. in terms of the inter-dependencies between all T and WP). Specifically, the proposed framework evaluates the comprehensibility by quantifying the cognitive (or conceptual) *complexity* at three different levels of granularity: *work product (WP), task (T),* and *methodology (M)* as a whole (Sub-characteristics SC1.1-SC1.3 respectively).

▪ ***Sub-characteristic SC1.1.*** **Cognitive Complexity - Work Product**

According to Rossi and Brinkkemper [51], the total conceptual complexity of an individual work product type[3] (WP) is reflected by the number of, and relationship between, its encapsulated objects, properties, and relationships (see Section 2.2.3). To this end, the relevant metrics defined in [51], see below, were integrated in the proposed evaluation framework as follows.

Measurement Specifics: *ratio-scale metrics (WP granularity)*

SC1.1:M1. ***n(O):*** **number of object types per WP type**

SC1.1:M2. ***n(P):*** **number of property types per WP type**

SC1.1:M3. ***n(R):*** **number of relationship types per WP type**

SC1.1:M4. ***P_o:*** **mean number of properties per object type**

SC1.1:M5. ***P_r:*** **mean number of properties per relationship**

SC1.1:M6. ***C':*** **total conceptual complexity of WP type** (derivative of *SC1.1:M1-M3,* see [51]).

▪ ***Sub-characteristic SC1.2.*** **Cognitive Complexity - Task**

The cognitive complexity of individual tasks (T) can be reflected by the number and complexity of the associated input/output work products (WP) and techniques (Te(T)). Specifically, this paper *proposes four dedicated metrics* (see below) for the purpose of task complexity quantification.

Measurement Specifics: *ratio-scale metrics (T granularity)*

SC1.2:M1. ***BTI:*** **basic task interaction,**

$$BTI = NWP_{in}(T) + NWP_{out}(T),$$

where NWP_{in} and NWP_{out} are the numbers of input and output work products manipulated (i.e. *interacted with*, as per meta-model from Section 2.1) by a task *T*. Although BTI can be easily calculated during the initial evaluation (or pre-screening) of a methodology, it represents a coarse estimation of the task complexity. In contrast, the following metric (WTI) requires greater measurement effort, but is more precise insofar as it considers the conceptual complexity of the involved WP types.

SC1.2:M2. ***WTI:*** **weighted task interaction,**

$$WTI = \sum_{wp_{in} \in WPin(T)} C'(wp_{in}) + \sum_{wp_{out} \in WPout(T)} (C'(wp_{out}) * F1),$$

where C'(wp) is the cognitive complexity of an input/output work product *wp* (see metric SC1.1:M7) multiplied by the (*optional*) *task uniqueness* factor *[F1]*. This factor is needed because the sets of input (WP_{in}) and output (WP_{out}) work product types manipulated by a given task can intersect (i.e. some, or even all, work product types can have both input and output roles within the context and scope of a given task). This can potentially reduce the cognitive effort needed to analyse intersecting sets of input and output work product types (compared to analyzing disjoint sets) given that the producer (P) will be analysing the structure and other relevant characteristics of intersecting work product types only once. In such cases, *the cognitive complexity of any output work product type* WP_{out} *that has a corresponding (i.e. same) input work product type* WP_{in} *should be artificially decreased* by a task uniqueness factor (F1) so as to provide more accurate assessment of task complexity. This is further illustrated in the example provided in Section 7.1. Note that at this stage the actual value of the uniqueness factor is set to the arbitrary value of 0.5 (i.e. *F1 = 0.5*). However, this value is provisional and should be evaluated empirically in future work.

SC1.2:M3. ***N(Te(T)):*** **number of supporting techniques for a task**

SC1.2:M4. ***WN(Te(T)):*** **weighted number of supporting techniques for a task,**

$$WN(Te(T)) = N(Te_{qn}(T)) + (N(Te_{ql}(T)) * F2),$$

where $N(Te_{qn}(T))$ is the number of associated quantitative task techniques; and $N(Te_{ql}(T))$ is the number of associated qualitative task techniques multiplied by the *technique weighting* factor *[F2]* so as to differentiate between the quantitative and qualitative nature of the provided techniques. This is because quantitative techniques are beneficial to the overall development process since they allow the producers (P) to perform tasks (T) and evaluate the instances of manipulated work products (WP) in an objective and uniform manner. Furthermore, quantitative techniques provide opportunities for process automation where the applicable work products can be auto-generated by supporting tools. In contrast, qualitative techniques can have a negative effect on the cognitive complexity of tasks, and can also result in a production of internally inconsistent work products given that the application of the tasks will be less predictable (i.e. it requires discretional judgment) [10]. To this end, *qualitative techniques should be weighted higher than quantitative ones* so to provide more accurate measure of task complexity, with the weight factor is presently set to the arbitrary value of 2 (i.e. *F2 = 2*). Similarly to factor F1, in SC1.2:M2, this value is provisional and should be evaluated in future work.

▪ ***Sub-characteristic SC1.3.*** **Cognitive Complexity - Methodology**

The complexity of a methodology is evaluated via three complementary measurement approaches:

- Using validated and empirically evaluated metrics proposed in FMESP [14] (see Section 2.2.3), which are designed to quantify the *structural properties of software development methodologies* based on the total number of

[3] Evaluating the complexity of a given work product type involves the (*internal*) assessment of its underlying meta-model (see Section 7.1). This is in contrast to the (*external*) complexity assessment of the work product instances resulted from applying a methodology in practice.

producers (P), work product types (WP), and tasks (T), as well as the inter-dependency between all T and WP [14]. Metrics *SC1.3:M1-M6* below are taken from FMESP.

- Deriving the *combined and mean cognitive complexity of all incorporated work products types and tasks* as quantified by [51] (see Sub-characteristic SC1.1) and the newly proposed metrics (Sub-characteristic SC1.2) respectively. See metrics *SC1.3:M7-M10* below.

- Quantifying the usage of of included quantitative (Te_{qn}) and qualitative (Te_{ql}) techniques using a set of *newly-proposed* metrics *SC1.3:M11-M14* below.

Measurement Specifics: *ratio-scale metrics (M granularity)*

SC1.3:M1. ***NP:*** **number of producers** [14]

SC1.3:M2. ***NWP:*** **number of work product types** [14]

SC1.3:M3. ***NT:*** **number of tasks** (adapted[4] from [14])

SC1.3:M4. ***$NDWPT_{in}$:*** **number of input dependencies of all WP with all T** (adapted from [14])

SC1.3:M5. ***$NDWPT_{out}$:*** **number of output dependencies of all WP with all T** (adapted from [14])

SC1.3:M6. ***NDWPT:*** **number of dependencies between all WP and all T** (adapted from [14])

$$NDWPT = NDWPT_{in} + NDWPT_{out}$$

SC1.3:M7. ***C' (M):*** **total WP complexity of a methodology [51],**

$$C'(M) = \sum_{wp \in WP} C'(wp),$$

where C'(wp) is the conceptual complexity of a given work product type *wp* (see metric SC1.1:M8).

SC1.3:M8. ***MWPC:*** **mean WP complexity of a methodology,**

$MWPC = C'(M)/NWP$, (see metrics SC1.3:M7 and SC1.3:M2).

SC1.3:M9. ***TTI:*** **total T interaction of a methodology,**

$$TTI = \sum_{t \in T} WTI(t),$$

where WTI(t) is the weighted task (t) interaction (see metric SC1.2:M2).

SC1.3:M10. ***MTI:*** **mean T interaction of a methodology,**

$MTI = TTI/NT$, (see metrics SC1.3:M9 and SC1:3:M3).

SC1.3:M11. ***NTe:*** **total number of (quantitative and qualitative) techniques**

SC1.3:M12. ***WNTe:*** **weighted number of techniques**

$$WNTe = \sum_{t \in T} WN(Te(t)),$$

where WN(Te(t)) is the weighted number of task (t) techniques (see metric SC1.2:M4).

SC1.3:M13. ***TTR:*** **technique to task ratio,**

$TTR = NTe/NT$, (see metrics SC1.3:M11 and SC1.3:M3).

SC1.3:M14. ***TQD:*** **technique quantification degree,**

$TQD = NTe_{qn} / NTe$,

where $N(Te_{qn})$ is the total number of quantitative techniques. Possible Values: [0 (total lack of quantitative techniques) to 1 (full technique quantification)]. Note that the ratio of qualitative techniques will be inversely proportional to TQD.

[4] FMESP utilised *activities* (A) as a target work unit type; however, using finer-grained *tasks* (T) could result in more precise measurement outcomes, and as such, the original metrics were updated accordingly.

V. Methodology Evaluation Framework - "Underlying Process" Perspective

This section covers the *process-related aspects* of a software development methodology. As was the case with Section IV, the logically related sub-characteristics (again denoted by the *SCn* marker in the following sub-sections) are grouped together and presented in terms of three high-level quality characteristics (denoted by *Cn*), *Completeness (C2), Efficiency (C3), and Effectiveness (C4)*. The *Completeness* characteristic is designed to be *tailored and customised for a particular development paradigm* as has been done in this section for a specific case of SOC. Also, the *Efficiency* and *Effectiveness* characteristics are evaluated using *external* metrics.

5.1 Characteristic C2. ***Completeness***

In this framework, completeness of a methodology is defined as the extent to which its underlying process is capable of providing paradigm-specific support for: i) the core development lifecycle phases (sub-characteristic SC2.1); ii) paradigm-specific modelling (SC2.2).; iii) relevant support technologies (or solution architectures) and standards (SC2.3); iv) verification and validation mechanisms (SC2.4); and v) project management activities (SC2.5). Note that the purpose of this characteristic is to outline an explicit approach to process evaluation and show examples of relevant features, rather than provide an exhaustive set of all possible features. Moreover, for brevity, only the features pertinent to Sub-Characteristics SC2.1 and SC2.2 are shown in this section, with the remaining features (covering Sub-Characteristics SC2.2-2.4) provided in Appendix C.

- ***Sub-characteristic SC2.1.*** **Lifecycle**

According to the *ISO/IEC 12207:2008 Systems and Software Engineering - Software life cycle processes* standard [23], a core software development process can be subdivided into a number of lifecycle phases, including Software Requirements Analysis (SRA), Software Architectural Design (SAD), and Software Detailed Design (SDD). To this end, this sub-characteristic is designed to evaluate development methodologies with respect to their *paradigm-specific* support of the SRA, SAD and SDD phases. In the case of SOC, the following mapping can be established and applied to guide the evaluation of the lifecycle support (in terms of the included tasks (T)) in the context of SOC:

(1) SRA <-> Service Identification (SI) [1] (or SO Discovery [8]);

(2) SAD <-> Service Specification (SS) [1] (or SO Analysis [8]);

(3) SDD <-> Service Realisation (SR) [1] (or SO Design [8]).

NOTE: The ISO/IEC 12207:2008 standard also includes a Software Coding and Testing phase (SCT) as one of the core development phases. This phase and associated activities (e.g. coding, and preparing and executing test cases) are not covered in the proposed framework. Firstly, the coding activities are highly dependent on the specific technology in use, and as such, the existing SO methodologies do not cover them in detail. Secondly, the ISO/IEC 12207:2008 standard recommends

independent verification and validation processes; therefore, methodological support for testing activities should be provided independently from the core development methodology.

Measurement Specifics: *ordinal-scale feature [1(no support) – 5 (full support)] (T granularity)*

SC2.1:F1. **Provides support for a core lifecycle task (T_x)**

NOTE: This feature should be applied (individually) to each and every core development task (T_x) included in the Service Identification (SI), Service Specification (SS), and Service Realisation (SR) phases (see Appendix B for an example list of common tasks extracted from existing literature).

- ***Sub-characteristic SC2.2.*** **Paradigm-Specific Modelling**

One of the main objectives of any software methodology is to provide comprehensive support for the fundamental design principles of a given development paradigm.

Measurement Specifics: *ordinal-scale features [1(strongly disagree) – 5(strongly agree)] (M granularity)*

SC2.2:F1. **Provides explicit coverage of different types of SO systems (e.g. SOS, PARSOS, PURSOS [46])**

SC2.2:F2. **Provides explicit coverage of different types of SO relationships (e.g. intra-service, indirect extra-service, and direct extra-service [46])**

Provides explicit coverage of different types of services according to their:

SC2.2:F3. **Purpose (e.g. process, task, entity, utility)**

SC2.2:F4. **Compositional aspect (e.g. composite and atomic)**

SC2.2:F5. **Functional scope/granularity (e.g. fine- and coarse- grained)**

SC2.2:F6. **Enforces the *service metamorphosis* (or a "metamorphosis embodiment") [2] in which service concept is consistently propagated throughout development phases and activities, and transformed from: *conceptual service -> analysis service -> design service -> solution service***

SC2.2:F7. **Promotes *loose-coupling* between services (e.g. provides explicit quantitative or/and qualitative techniques (Te) for avoiding direct-extra service relationships [48]**

SC2.2:F8. **Incorporates techniques for evaluating and managing *service granularity* [2]**

Provides methodological support for the core tasks needed to define, publish, and maintain service contracts, including:

SC2.3:F9. **Definition of service contracts using formats prescribed by the registry in use (i.e. project and infrastructure dependent)**

SC2.3:F10. **Publication and maintenance of service contracts in *external* registries (e.g. UDDI) [59]**

SC2.3:F11. **Publication and maintenance of service contracts in *internal* registries (e.g. WSO2 Governance Registry http://wso2.com/products/governance-registry/)**

Provides methodological support for the major service discovery and integration tasks, including:

SC2.2:F12. **[At the *provider* level] Documenting service interfaces using semantic languages (e.g. SWSL [8])**

SC2.2:F13. **[At the *consumer* level] *Static* evaluation, selection, and integration of external SaaS offerings [2]**

SC2.2:F14. **[At the *consumer* level] *Dynamic* discovery, selection, and integration of external SaaS offerings [45]**

- ***Sub-characteristic SC2.3.*** **Support Technologies and Standards**

Supporting contemporary paradigm-specific technologies and standards can potentially increase the practical applicability of a software development methodology [10]. In the case of SOC, such technologies and standards can include, but are not limited to, ESB architecture [5], internal and external service registries [59], and various ws* standards [8]. Refer to Appendix C for the list of corresponding features.

- ***Sub-characteristic SC2.4.*** **Verification and Validation**

Evaluating architectural correctness (or *verifiability*) of the system using formal verification analysis techniques[5] can minimise future reliability issues [10]. Also, according to the ISO/IEC 12207:2008 *Systems and Software Engineering - Software life cycle processes* standard [23], a development process should encourage architectural consistency (or *validity*) via the means of *requirements traceability* and *design consistency* [12].

- Refer to Appendix C for the list of corresponding features.

- ***Sub-characteristic SC2.5.*** **Project Management**

Providing paradigm-specific support for *quality management*, and *effort and cost estimation* activities can encourage the production of quality software products in line with the available resources and timelines [35]. Also, formal project planning tasks can support an *incremental and iterative development process*, which is recommended by the existing software engineering standards and methodologies [10, 12, 23, 29], and is of particular significance to SOA-based projects due to its support for: i) continuous integration and verification/validation of the evolving services ecosystem; ii) early delivery of capability subsets (i.e. collection of services); and iii) early detection and mitigation of defects. Refer to Appendix C for the list of corresponding features.

- ***Sub-characteristic SC2.6.*** **Total Completeness**

This sub-characteristic quantifies overall process completeness of a methodology according to sub-characteristics SC2.1-SC2.5.

Measurement Specifics: *ratio scale metrics (M granularity)*

SC2.6:M1. ***LSC:*** **lifecycle coverage,**

LSC = NST / NIT,

where NST and *NIT* are the numbers of supported and included (i.e. as per Appendix B) lifecycle tasks (T) respectively. An *expected* task is deemed to be *supported* according to the values assigned to the corresponding feature SC2.1:F1, see Note

[5] The verification analysis is done by providing a formal proof on an abstract mathematical model, the correspondence between the model and the nature of the system being otherwise known by construction [4].

below. Possible Values: [0 (total lack of lifecycle coverage) to 1 (totally covered)].

NOTE: The decision was made to quantify the lifecycle coverage, as well as the other coverage related metrics shown in this and the next sections, by calculating the percentage of the "agree" (the value 4) and "strongly agree" (the value 5) feature responses, instead of calculating the arithmetic mean of all the responses. This was done in order to satisfy one of the key principles of measurement theory and scale types [57] in which ordinal scale data (e.g. Likert item used in SC2.1:F1) can only be tested for equality and order (via formal relations '=', '≠', '<', and '>'), but should not be used to produce mean values across the available data range.

SC2.6:M2. ***PSC:*** **paradigm-specific support coverage,**

PSC = NSF / NIF,

where NSF is the number of supported features as reflected by the evaluation scores of 4 ("agree") or 5 ("strongly agree"); and *NIF* is the total number of features included in sub-characteristic SC2.2 (e.g., currently, *NIF=14*).

SC2.6:M3. ***STC:*** **support technologies coverage,**

STC = NSF / NIF,

where NSF and *NIF* are the numbers of supported and included features of sub-characteristic SC2.3.

SC2.6:M4. ***VVC:*** **verification and validation coverage,**

VVC = NSF / NIF,

where NSF and *NIF* are the numbers of supported and included features of sub-characteristic SC2.4.

SC2.6:M5. ***PMC:*** **project management coverage,**

PMC = NSF / NIF,

where NSF and *NIF* are the numbers of supported and included features of sub-characteristic SC2.5.

5.2 Characteristic C3. ***Efficiency***

In this framework, efficiency of a methodology is defined as the extent to which the productivity of the producers (P) is increased. Improving the process efficiency is considered to be one of the core objectives of any software development methodology [23].

- ***Sub-characteristic SC3.1.*** **Developer productivity**

Measurement Specifics: ***external*** *ratio-scale metric (T granularity)*

SC3.1:M1. ***PE:*** **producer efficiency,**

$$PE = \sum_{t \in T(sp)} (Time\ (t) / WPi_{out}(t)),$$

where Time (t) is a total time in minutes spent on performing each task *t* needed to produce a complete software product *sp*; and WPi_{out} *(t)* is a total number of individual work product instances produced by *t*. The lower value of PE indicates higher efficiency. Note that it is important to ensure that the PE is measured for comparably sized (i.e. in terms of functional and non-functional requirements) systems when analysing and comparing values obtained for different methodologies. Also, the quality and complexity of the produced work products should be taken into consideration.

5.3 Characteristic C4. ***Effectiveness***

In this framework, effectiveness of a methodology is defined as the extent to which its underlying process supports the production of quality software products that meet user requirements and other pre-defined functional and non-functional constraints. Similar to process efficiency, improving the development effectiveness is considered to be one of the core objectives of any software development methodology [23].

- ***Sub-characteristic SC4.1.*** **Work Product Quality**

The (external) quality characteristics (i.e. non-functional properties) of a software product (e.g. reliability and maintainability) are shown to be affected by its structural properties (e.g. size, coupling, and cohesion) [16], and as such, it is beneficial to assess the quality of work product instances at both *external* and *internal* (i.e. structural) levels.

Measurement Specifics: ***external*** *ratio-scale metrics (WP granularity)*

SC4.1:M1. ***EQ:*** **external quality of a produced product,**

$$EQ = \sum_{wp \in WPi_{out}} (ExQualityFactor(wp)),$$

where ExQualityFactor(*wp*) is a measurement of a given quality characteristic of a produced work product instance $wp \in WPi_{out}$ as measured by existing ISO/IEC:9126 metrics. The evaluator should select target quality characteristic (e.g. efficiency, portability, and reliability) based on the evaluation goals and requirements.

SC4.1:M2. ***IQ:*** **internal (structural) quality of a produced product,**

$$IQ = \sum_{wp \in WPi_{out}} (InQualityFactor(wp)),$$

where InQualityFactor(*wp*) is a measurement of a given structural property of a produced work product instance $wp \in WPi_{out}$ as measured by existing SOC-specific software metrics (e.g. [47-48]). Note that the evaluator should select target structural properties (e.g. coupling, cohesion, complexity, and size) based on the evaluation goals and requirements.

VI. Methodology Evaluation Framework – "Deliverable Product" Perspective

This section presents quality characteristics, sub-characteristics, and associated features and metrics designed to evaluate inherent *product characteristic*s of a methodology, such as its *maturity* (Characteristic C5), *usability* (C6), and *availability* (C7). Note that all proposed product-related features/metrics are of the internal type.

6.1 Characteristic C5. Maturity

In this framework, maturity of a methodology is defined as the extent to which its underlying model has been fully developed (or matured) insofar as its underlying model (i.e. tasks (T) and work product types (WP)): i) is standardised (Sub-characteristic SC5.1); ii) is stable (i.e. does not change frequently) (SC5.2); and iii) is indicated to be mature by the *miscellaneous maturity* metrics (SC5.3*).* The maturity is an important characteristic of any methodology given that frequent changes to the methodology's underlying model can *attract high costs associated with producer re-training and tool updates* [10].

- ***Sub-characteristic SC5.1.*** **Standardisation**

While not immediately mature, standardised tasks (T)

and work products (WP) can potentially reflect the maturity of a methodology, and can also result in general quality improvements [33].

Measurement Specifics: *ratio-scale metric (M granularity)*

SC5.1:M1. ***SD:*** **standardisation degree,**

SD = (NST/NT + NSWP/NWP) / 2,

where NST and *NSWP* are the numbers of standardised (e.g. by relevant standardisation bodies such as the IEEE, ISO/IEC, OASIS, and OMG) tasks $T_{st} \subseteq T$ and work product types $WP_{st} \subseteq WP$ respectively; and *NT* and *NWP* are defined as part of Sub-characteristic SC1.3. Possible Values: [0 (total lack of standardisation) to 1 (total standardisation)].

- ***Sub-characteristic SC5.2.*** **Stability**

Stability evaluates the degree of change in between methodology releases where a stable methodology is said to exhibit a low degree and frequency of release change.

Measurement Specifics: *ratio-scale metrics (M granularity)*

SC5.2:M1. ***LRCD:*** **latest release change degree,**

LRCD = (NCT/NT + NCWP/NWP) / 2,

where NCT and *NCWP* are the numbers of changed tasks $T_{ch} \subseteq T$ and work product types $WP_{ch} \subseteq WP$ respectively; and *NT* and *NWP* are defined as part of Sub-characteristic SC1.3. Possible Values: [0 (total lack of change in a given release) to 1 (total change (i.e. a complete replacement))].

SC5.2:M2. ***MCD:*** **mean change degree,**

$$MCD = \sum_{i=1}^{NRV} LRCD_i / NRV,$$

where RCD is defined in SC5.2:M1; and *NRV* is the total number of all released versions of the methodology (see SC5.3:M2). Possible Values: [0 (total lack of change in between releases i.e. high stability) to 1 (total change in between each release i.e. low stability)].

- ***Sub-characteristic SC5.3.*** **Miscellaneous Maturity**

This sub-characteristic covers miscellaneous properties of maturity not captured by the previous sub-characteristics.

Measurement Specifics: *ratio-scale metrics (M granularity)*

SC5.3:M1. ***NYSR:*** **number of years since the first public release of the methodology**

SC5.3:M2. ***NRV:*** **number of released versions of the methodology**

6.2 Characteristic C6. Usability

In this framework, usability is defined as the extent to which the methodology can be learned and applied by the users. Specifically, the proposed framework aims to quantify the *usability*, from perspective of producers, of three main (physical) deliverables of any software development methodology - *documentation, supporting software tools, and training* (Sub-characteristics SC6.1-SC6.3 respectively).

- ***Sub-characteristic SC6.1.*** **Documentation**

An appropriate (e.g. consistent, comprehensible, and comprehensible) documentation can promote systematic application of a methodology by a broad spectrum of potential producers (P) [28]. The documentation itself can be classified into *formal* and *informal* types. Formal documentation includes formal grammars, transformation algorithms, process algebra definitions, graphical notation, etc. Informal documentation can take a form of natural language descriptions, illustrations (e.g. screen captures, diagrams, figures/tables), and practical examples or case studies.

Measurement Specifics: *ordinal-scale features [1(strongly disagree) – 5(strongly agree)] (T and WP granularity)*

- Provides appropriate documentation for a task (T)/work product type (WP)

SC6.1:F1. **The documentation is consistent**

SC6.1:F2. **The documentation is comprehensive**

SC6.1:F3. **The documentation is comprehensible**

SC6.1:F4. **The documentation is technology-neutral**

SC6.1:F5. **The documentation is appropriately illustrated**

SC6.1:F6. **The documentation provides realistic examples**

SC6.1:F7. **Formal documentation affords[6] its purpose**

NOTE: The above features should be applied individually to each and every T and WP included in the methodology, and can be readily decomposed and redefined for specific formal and informal documentation types on an as needed basis (e.g. SC6.1:F1 can be redefined as <*Graphical notation* is consistent> etc.).

- ***Sub-characteristic SC6.2.*** **Tool Support**

Integrated tool support (e.g. IDEs, CASE and project management and quality assurance tools) can increase productivity of the producers (P) (e.g. by supporting automated WP transformations and updates) and encourage task (T) and work product (WP) consistency. To this end, tool support is considered to be an integral part of any software development methodology [27].

Measurement Specifics: *ordinal-scale features [1(strongly disagree) – 5(strongly agree)] (T granularity)*

- Provides appropriate software tool support for a task (T)

SC6.2:F1. **The tool support is consistent**

SC6.2:F2. **The tool support is comprehensive**

SC6.2:F3. **The tool support is comprehensible**

SC6.2:F4. **The tool support includes realistic examples**

SC6.2:F5. **Tool support affords[6] its purpose**

SC6.2:F6. **Tool support is usable according to *chosen usability assessment techniques*.** Note: the choice of a particular technique should be dependent on the evaluator's current expertise, existing organisational practices, and available evaluation resources. As an example, the following [37] usability heuristics can be applied:

- **User retention**
- **User control and freedom**
- **Consistency and standards**
- **Error prevention**

[6] A (perceived) *affordance* is the design aspect of an object that suggests how it might be used i.e. a visual representation of a "properly" designed object should provide an immediate clue to its function [38].

- **Recognition rather than recall**
- **Flexibility and efficiency of use**
- **Aesthetic and minimalist design**
- **Help users diagnose, and recover from errors**
- **Help and documentation**

▪ ***Sub-characteristic SC6.3.*** **Training**

Dedicated training can improve the methodology's learning process, thereby increasing its usability [55].

Measurement Specifics: *ordinal-scale features [1(strongly disagree) – 5(strongly agree)] (T granularity)*

SC6.3:F1. **On-site training is provided, by the methodology provider or relevant third parties, for a task (T)**

SC6.3:F2 **Public workshops are conducted for a task (T)**

SC6.3:F3. **Self-guided training is available for a task (T)**

SC6.3:F4. **Formal certification is provided for a task (T)**

NOTE: The above features should be applied individually to each and every T included in the methodology, and can be refined to evaluate three important aspects of the above training modes - consistency, comprehensiveness, and comprehensibility (as was the case with Sub-characteristics SC6.1 and SC6.2).

▪ ***Sub-characteristic SC6.4.*** **Total Usability**

This sub-characteristic reflects overall (i.e. documentation, tool support, and training) usability coverage of a methodology according to Sub-characteristics SC6.1-SC6.3.

Measurement Specifics: *ratio scale metrics (M granularity)*

SC6.4:M1. ***DUC:*** **documentation usability coverage**

DUC = (Count(NDT)/NT + Count(NDWP)/NWP) / NSF / 2,

where NT and *NWP* are defined as part of Sub-characteristic SC1.3; *NSF* is the total number of evaluation features of SC6.1 (currently NSF=7); and *Count (NDT)* and *Count* (*NDWP)* are the counts of all positive responses indicating appropriate documentation, for a given T or WP respectively, as reflected by the values 4 or 5 assigned to the corresponding (sub) features SC6.1:F1-F7. Possible Values: [0 (total lack of documentation) to 1 (totally documented)].

SC6.4:M2. ***TSUC:*** **tool support usability coverage**

TSUC = Count (NST) / NT / NSF,

where NT is defined as part of Sub-characteristic SC1.3; *NSF* is the total number of evaluation features of SC6.2 (currently NSF=6); and *Count (NST)* is the count of all positive responses indicating appropriate tool support as reflected by the values 4 or 5 assigned to features SC6.2:F1-F6. Possible Values: [0 (total lack of appropriate tool support) to 1 (totally supported) by appropriate tools].

SC6.4:M3. ***TUC:*** **training usability coverage**

TUC = Count (NTT) / NT / NSF,

where NT is defined as part of Sub-characteristic SC1.3; *NSF* is the total number of evaluation features of SC6.3 (currently NSF=4); and *Count (NTT)* is the count of all positive responses indicating appropriate training as reflected by the values 4 or 5 assigned to features SC6.3:F1-F4. Possible Values: [0 (total lack of appropriate training) to 1 (totally supported by training)].

6.3 Characteristic C7. ***Availability***

In this framework, availability is defined as the extent to which the methodology can be openly and/or freely accessed. Similarly to the usability characteristic, the *availability* of a methodology is quantified based on the availability of its *documentation*, *tool support*, and *training* (Sub-characteristics SC7.1-SC7.3 respectively). Assuming all other quality characteristics (i.e. Characteristics C1-C6) equal, higher availability is desirable [36].

▪ ***Sub-characteristic SC7.1.*** **Documentation**

Measurement Specifics: *ordinal-scale features [1(strongly disagree) – 5(strongly agree)] (M granularity)*

SC7.1-F1: **The methodology's documentation is available in the open or public domain[7] (e.g. creativecommons.org)**

SC7.1-F2: **The documentation upgrades are provided free of charge**

▪ ***Sub-characteristic SC7.2.*** **Tool Support**

Measurement Specifics: *ordinal-scale features [1(strongly disagree) – 5(strongly agree)] (M granularity)*

SC7.2-F1: **The methodology's tool support is available in the public domain (e.g. under GNU initiative)**

SC7.2-F2: **The tool support upgrades are provided free of charge**

▪ ***Sub-characteristic SC7.3.*** **Training**

Measurement Specifics: *ordinal-scale features [1(strongly disagree) – 5(strongly agree)] (M granularity)*

SC7.3-F1: **On-site training is provided free of charge**

SC7.3-F2: **Public workshops are offered free of charge**

SC7.3-F3: **Self-guided training resources are available in the public domain (e.g. creativecommons.org)**

SC7.3-F4: **Formal certification is provided free of charge**

▪ ***Sub-characteristic SC7.4.*** **Total Availability**

This sub-characteristic reflects overall (i.e. documentation, tool support, and training) availability coverage of a methodology according to Sub-characteristics SC7.1-SC7.3. Note that for brevity the actual metric calculation formulas and details have been omitted since they follow the same pattern as the previous coverage metrics (e.g. SC6.4:M1-M3).

Measurement Specifics: *ratio scale metrics (M granularity)*

SC7.4-M1: **DAC: documentation availability coverage**

SC7.4-M2: **TSAC: tool support availability coverage**

SC7.4-M3: **TAC: training availability coverage**

VII. Practical Application

An exploratory *case study* was conducted to demonstrate practical applicability [27] of the proposed framework, and outline directions for future work. The case study involved the evaluation of a sub-set of the *Service-oriented Modelling and Architecture (SOMA)* [1] methodology using a sub-set of the proposed ordinal-scale features and ratio-scale metrics. SOMA is an end-to-end software development methodology for analysing,

[7] Open licensing model is common in ICT with a number of initiatives providing free and open access to software products and other works (e.g. http://www.gnu.org/ and http://creativecommons.org).

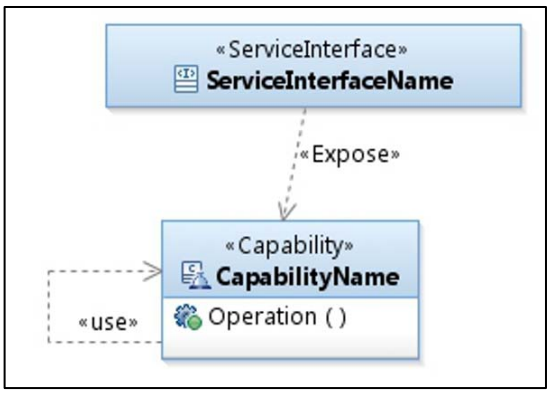

Fig. 3 SoaML service dependencies diagram – meta-model

designing and building SOA-based solutions, which was initially proposed in 2004, and has since undergone three major revisions. SOMA has comprehensive documentation and tool support, and was chosen since it was readily available (via IBM Academic Initiative), and the present authors had practical experience with its application (see [47-48]).

The goal of this case study is to demonstrate the application of selected parts of the proposed framework (i.e. provide a meaningful exemplar), rather than conduct a comprehensive evaluation of either SOMA or framework as a whole. Specifically, this section demonstrates the process of evaluating Sub-characteristic SC1.2 (*Cognitive Complexity - Task*) using two ratio-scale complexity metrics (in Section 7.1), and also provides a brief example of feature analysis (in Section 7.2), using eleven proposed features, as applied to Sub-characteristic SC2.2 (*Paradigm-Specific Modelling*).

7.1 Metrics based assessment

The SOMA task, *"identification of service capabilities from business processes"*, is part of the Service Identification phase and was selected for the purpose of demonstrating the quantification procedure of the cognitive complexity of a task, including the application of relevant weight factors. This particular task was chosen because it is of manageable size and provides coverage of all structural constructs needed to calculate the task complexity values. Specifically the following constructs are covered:

***<Input> work product* types** (WP_{in}):

- BPMN process model

- Service dependencies diagram (from SoaML[8])

***<Output> work product* types** (WP_{out})*:*

- Service dependencies diagram

<***Quantitative*> *techniques*** (TE_{qn}):

- Decompose a given business process into 3 levels of decomposition, process, sub-process, leaf-level sub-process

<***Qualitative*> *techniques*** (TE_{ql}):

- Eliminate highly abstract processes

- Eliminate fine-grained tasks

Metrics Calculation:

SC1.2:M2. ***WTI:* weighted task interaction,**

$$WTI = \sum_{wp_{in} \in WPin(T)} C'(wp_{in}) + \sum_{wp_{out} \in WPout(T)} (C'(wp_{out})*F1),$$

The first step in calculating the value of WTI involves determining the conceptual complexity of all manipulated input ($C'(wp_{in})$) and output ($C'(wp_{out})$) work product types, with this particular task having two input WP types (BPMN and ServiceDependenciesDiagram) and one output WP type (ServiceDependenciesDiagram):

- The conceptual complexity of the BPMN work product type has been previously evaluated in [50], with the value of $\mathbf{C'(_{BPMN})} = \mathbf{93.6}$;

- To calculate the conceptual complexity of the ServiceDependenciesDiagram work product type, the meta-model of which is depicted graphically in Fig. 3, all possible objects (O), properties (P), and relationships (R) were identified as follows:

O= {capability, service interface}

P= {capability name, capability operation, operation qualifier, service interface name}

R= {use, expose}

According to the above, the work product complexity values for the ServiceDependenciesDiagram can then be calculated as:

SC1.1:M1. $n(O_{ServiceDependenciesDiagram}) = 2$

SC1.1:M2. $n(P_{ServiceDependenciesDiagram}) = 4$

SC1.1:M3. $n(R_{ServiceDependenciesDiagram}) = 2$

SC1.1:M4. $P_o\,(_{ServiceDependenciesDiagram}) = 2$

SC1.1:M5. $P_r\,(_{ServiceDependenciesDiagram}) = 2$

SC1.1:M6. $\mathbf{C'(_{ServiceDependenciesDiagram})}$=√(n(O)^2+n(P)^2+n(R)^2) = **4.9**

It can be observed, that the total conceptual complexity of a service dependencies diagram (C' = 4.9) is much lower than the complexity of BPMN (C' = 93.6). This was expected given that a service dependencies diagram covers a limited number of objects, properties, and relationships compared to BPMN. This is also in line with the previous complexity calculations for simpler work product types evaluated by other researchers [54] (e.g. UML class (C' = 26.4) and UML component (C' = 15.65) diagrams).

The (total) input and output work product complexity can then be calculated as:

$\sum C'(wp_{in}) = C'(_{BPMN}) + C'(_{ServiceDependenciesDiagram}) = 93.6 + 4.9 =$ **98.4**

and

$\sum C'(wp_{out})*F1 = C'(_{ServiceDependenciesDiagram})*0.5 =$ **2.45**

Note that the (optional) *weight* factor <*F1=0.5*> was applied since a ServiceDependenciesDiagram is used as both an input and output work product type.

Finally, the total weighted task interaction (WTI) for the SOMA task, *"identification of service capabilities from business processes"* is: WTI = 98.4 + 2.45 = **100.85**

Note that since the aim of this study is to demonstrate the application of selected parts of the framework (rather than conduct a complete evaluation of SOMA), we do not draw any comparative or relative conclusions regarding the magnitude of this complexity value.

[8] SoaML is an open source specification project from the Object Management Group, describing a UML profile and meta-model for the modelling and design of SO systems (www.omg.org/spec/SoaML).

SC1.2:M4. ***WN(Te(T))*** **– weighted number of supporting techniques for a task,**

$WN(Te(T)) = N(Te_{qn}(T)) + (N(Te_{ql}(T))*F2)$,

The SOMA task, *"identification of service capabilities from business processes"* is supported by one quantitative (Te_{qn}) and two qualitative (Te_{ql}) techniques. To this end, the value of $WN(Te(T))$ can be calculated as:

(1) $N(Te_{qn}) = \mathbf{1}$

(2) $N(Te_{q1})*F2 = 2*2 = \mathbf{4}$

Note that the *weighting* factor <*F2=2*> was applied to the *qualitative* techniques as was explained in Section IV.

(3) Finally, the weighted number of supporting techniques (WN(Te(T))) for the SOMA's *"identification of service capabilities from business rocesses"* task is: $WN(Te(T)) = 1+4 = \mathbf{5}$

Again, as with WTI, no conclusions about the relative magnitude of this value can be drawn.

7.2 Feature analysis

Providing a complete feature analysis for all the proposed feature-based sub-characteristics is beyond the scope of this paper. Instead, this section demonstrates the application of feature analysis to the arbitrary chosen sub-characteristic "Paradigm-specific Modelling" and its corresponding features (SC2.2:F1-F11). The evaluation results, ranked by the first author and shown in Table 2, suggest that overall SOMA appears to provide strong support for the fundamental concepts of SOC (as reflected by features SC2.2:F1-F11). However, SOMA *lacks methodological support for the service discovery* tasks (e.g. integration of semantic languages, and static and dynamic service discovery, selection, and integration) as reflected by the low ranks assigned to features SC2.2:F12-F14 (highlighted in bold). This in turn suggests the area of *possible methodological enhancements* that can be applied to SOMA.

TABLE 2.
EXISTING METHODOLOGY EVALUATION APPROACHES – SUMMARY

SUB-CHARACTERISTIC ID	FEATURE ID	RANK VALUE *1 2 3 4 5*
SC2.2. Paradigm-Specific Modelling	SC2.2:F1	4
	SC2.2:F2	3
	SC2.2:F3	4
	SC2.2:F4	5
	SC2.2:F5	5
	SC2.2:F6	4
	SC2.2:F7	3
	SC2.2:F8	4
	SC2.2:F9	4
	SC2.2:F10	3
	SC2.2:F11	3
	SC2.2:F12	**1**
	SC2.2:F13	**2**
	SC2.2:F14	**1**

VIII. CONCLUSION AND FURTHER RESEARCH

This paper presented a comprehensive and novel analytical framework for evaluating SO development methodologies using a set of qualitative features and combination of existing and newly-derived quantitative ratio-scale metrics structured in terms of a hierarchical quality model covering three unique perspectives of any generic software development methodology: i) the "structured artefact" perspective that treats a methodology as a *constructed entity structured in terms of a collection of process classes and their relationships*; ii) the "underlying process" perspective designed to be *tailored and adapted for a particular development paradigm* as has been done in this paper for the specific case of SOC; and iii) the "deliverable product" perspective, which treats a methodology as a *physical (deliverable) product* available in the form of documentation, supporting software tools, and training.An explorative case-study was presented in Section VII to demonstrate the practical applicability of the framework by applying a sub-set of the proposed metrics and features to a sub-set of a chosen SO development methodology, SOMA [1]. Specifically, the metric-based assessment illustrated the process of task complexity quantification, which can be replicated when measuring other methodological tasks and work products included in SOMA and other SO-specific methodological approaches (e.g. [2, 8, 43]). Also, it was shown that SOMA lacks methodological support for the service discovery tasks, thereby suggesting an area of possible methodological enhancements. In the future, it is envisioned that significant stakeholders will apply the proposed framework to perform an exhaustive *evaluation* and *comparison* of existing and/or newly-derived methodologies. For example, process engineers in commercial environments can apply the framework in order to gain an understanding of particular *strengths and weaknesses* of a methodology so as to perform an informed methodology enhancement. Also, the methodology users will be able to *compare* competing offerings, and *select the best method* according to specific project requirements and constraints.

Additionally, in future work, some of the currently proposed features could be replaced with *ratio-scale metrics* in order to provide a more objective evaluation mechanism. Also, the possibility of *extending the framework with ontological constructs* (see Section 2.2.2), so as to provide a more formal alternative to feature analysis, could be explored. For example, an ontology of SO development constructs could be derived and integrated into the proposed framework together with the concepts from the Function-Behavior-Structure (FBS) approach [30].

APPENDIX A. A META-MODEL OF A SOFTWARE DEVELOPMENT METHODOLOGY - DEFINITIONS

DEFINITION 1: Development Methodology **(M)**
A model of the software development process that consists of a set of selected, tailored, and integrated lifecycle phases, work products, work units, and associated methodology producers.

DEFINITION 2. Work Unit **(WU)**
An abstract methodology component that models an executable process operation.

D2.1. *Activity* ***(A)*** – a high-level structural work unit that encapsulates a cohesive set of tasks (T), and also defines their workflow sequencing. Note that a given task can be performed as part of different activities (resulting in many-to-many relationship between the activities and tasks). Example: Modelling business services.

D2.2. *Task* ***(T)*** – a core methodology component representing a low-level functionally cohesive operation performed by one or more producers for the purpose of manipulating (i.e. producing, modifying, or reviewing) one or more related work products (WP). Example: Identification of candidate business service operations.

D2.3. Technique **(Te(T))** – is an abstract core work unit that models the way of performing a task (T). The techniques can be classified into *quantitative* or *qualitative* types. Examples: Service cohesion metrics (quantitative); Informal guidelines for analysing the cohesiveness of service operations (qualitative).

DEFINITION 3. Producer **(P)**
A core methodology component that models (i.e. defines a set of required expertise (E)) an entity that performs one or more tasks (T). The producers can take a *primary or additional* role when performing a given task/s. Also, a producer can be either a human participant or a dedicated software tool that performs automated manipulation of work products. Example: Software Architect.

DEFINITION 4. Work Product **(WP)**
A core methodology component that represents anything of value that is *produced, modified, or reviewed* during the performance of one or more tasks (T). The work products can be categorised as either *mandatory* or *optional* in the context of a particular task/s. Also, the work products can support both *input* and *output* interactions with a task (T). Example: SRS document.
NOTE: a collection of work products (WP_i) is said to be associated with a task (T_i) only if WP_i is manipulated concurrently by T_i.

DEFINITION 5. Work Product Set **(WPS)**
A structural component that encapsulates a cohesive set of work products (WP) related by a lifecycle phase (LP). Example: Business process models and services.

DEFINITION 6. Lifecycle Phase **(LP)**
A structural component representing major logical partition of the development process. Lifecycle phases provide a natural organisation and timing to the performance of activities (A). Phases can be interleaved, overlapped, and iterated. Example: Software Requirements Analysis.

APPENDIX B. SERVICE-ORIENTED DEVELOPMENT LIFECYCLE - EXAMPLE TASKS

An example list of core SO development tasks (T), extracted and amalgamated from existing major SO methodological approaches (e.g. [1-2, 8, 43]), is shown below. Note that the description of the individual tasks is beyond the scope of this paper.

Service Identification Phase

- SI.T1: Goal-Service Modelling [1]
  - Identification of high-level business goals, and their refinement into sub-goals that must be met in order to fulfil the parent goal [1]
  - Identification of Key Performance Indicators (KPI) and metrics for sub-goals [1]
- SI.T2: Business Process Modelling [45]
  - Design of short-lived (sync) and long-lived (async) transaction activities [40]
  - Design of concurrent/parallel transaction activities [40]
  - Definition of business process orchestration and choreography specifications [45]
- SI.T3: Existing Asset Analysis [1]
- SI.T4: Domain Decomposition [1]
  - Identification of service capabilities from business process models [1]
- SI.T5: Candidate Service identification [1] from:
  - business goals and sub-goals [1]
  - business rules [1]
  - business processes [45]
  - domain models [1]
  - functional models [8]
  - existing assets [1]

Service Specification Phase

- SS.T1: Integration of non-functional requirements into service ecosystem [1]
- SS.T2: Identification and integration of security patterns [2]
- SS.T3: Specification of service *orchestration layer* [8]
- SS.T4: Specification of *business (or task) service layer* [8]
- SS.T5: Specification of *application service layer* [8]
- SS.T6: Refinement of application service layer into *entity and utility layers* [8]
- SS.T7: Service Litmus Test [1] (i.e. applying a set of design criteria to determine which candidate services should be exposed externally)
- SS.T8: Application of Service Analysis operations [2]:
  - Service *aggregation*: combining fine-grained services into a larger coarse-grained service [2]
  - Service *decomposition*: partitioning coarse-grained services into a collection of smaller fine-grained services [2]
  - Service *unification*: merging services with comparable attributes and business commonalities in order to avoid functional overlap/redundancy [2]
  - Service *subtraction*: reducing the scope of composite and atomic services (i.e. eliminating unnecessary functionality) [2]
- SS.T9: Specification of service components (i.e. service implementation elements) [1]

Service Realisation Phase

- SR.T1: Identification and documentation of service realisation decisions (e.g. in-house development, transformation, subscription, outsourcing) [1]
- SR.T2: Technical feasibility exploration [1]
- SR.T3: Service contract design including non-functional specification of service contracts[9] (i.e. SLAs, QoS) [31]
- SR.T4: Design of component-level interactions with the run-time middleware infrastructure (e.g. ESB) [1]
- SR.T5: Composite (i.e. orchestrated task) service layer design [2, 8]
- SR.T6: Design of *task service layer* internals (i.e. design of service implementation components and their run-time relationships and collaborations) [8]
- SR.T7: Design of *entity service layer* internals [8]
- SR.T8: Design of *utility service layer* internals [8]
- SR.T9: Definition of interfaces between service components and other operational (i.e. legacy) system layers [1]

APPENDIX C. PROCESS COMPLETENESS – EXAMPLE FEATURES FOR SUB-CHARACTERISTICS SC2.3-SC2.5

This appendix provides a list of features pertinent to Sub-characteristics SC2.3-SC2.5 (and for brevity omitted from the main body of the paper). Note that the included features are not exhaustive and could be extended and refined on an as needed basis. For example, the list of features provided for Sub-characteristic SC2.5 ("Project Management") could be augmented with additional features covering important management activities of project governance, reporting, and risk management.

Sub-characteristic SC2.3. **Support Technologies and Standards**

Measurement Specifics: *ordinal-scale features [1(no support) – 5(full support)] (M granularity)*

Provides support for the basic tasks needed to integrate the developed service ecosystem into ESB, including:

SC2.3:F1. **Registration of provided services in the ESB repository**

SC2.3:F2. **Definition of ESB-driven service communication protocols**

SC2.3:F3. **Creation of dedicated message and resource *routing* services, and definition and configuration of the routing rules**

Promotes the application of the core W3C standards (ws*), including:

SC2.3:F7. **WS-Coordination**

SC2.3:F8. **WS-Transaction**

SC2.3:F9. **WS-Reliable Messaging**

SC2.3:F10. **WS-Security**

SC2.3:F11. **WS-Policy**

SC2.3:F12. **WS-Addressing**

[9] A service contract consists of a general description (e.g. version, owner), a functional specification (service operations including inputs and outputs), a non-functional specification (e.g. QoS, SLAs), and services dependencies map (list of "used" and "used by" services).

Sub-characteristic SC2.4. **Verification and Validation**

Measurement Specifics: *ordinal-scale features [1(no support) – 5 (full support)] (M granularity)*

Incorporates formal system verification analysis techniques typical to the SO development, including:

SC2.4:F1. **Petri nets-based approaches**

SC2.4:F2. **FSM-based approaches**

SC2.4:F3. **Process algebra-based approaches**

SC2.4:F4. **Alternative verification approaches (i.e. not listed above)**

SC2.4:F5. **Enforces requirements traceability throughout the core phases of the SO development lifecycle**

i.e. WPS (SI) -> WPS (SS) -> WPS (SR),
where '->' means traceable

SC2.4:F6. **Enforces design consistency throughout the core phases of the SO development lifecycle**

i.e. WPS (SI) <-> WPS (SS) <-> WPS (SR)
where '<->' means internally consistent

Sub-characteristic SC2.5. **Project Management**

Measurement Specifics: *ordinal-scale features [1(no support) – 5(full support)] (M granularity)*

SC2.5:F1. **Uses existing SOC-specific effort and cost estimation frameworks (e.g. [39])**

SC2.5:F2. **Integrates with the ISO/IEC 25000:2005 [22] quality management standard in order to model, define, and manage functional and non-functional quality requirements**

SC2.5:F3. **Supports early quantification, using SOC-specific software metrics [47-48], of the *internal* quality characteristics (e.g. service cohesion) of the produced WP**

SC2.5:F4. **Supports the application of *external* quality metrics from the ISO/IEC 9126:1-4 suite of standards [19-21]**

Provides support for an *incremental* and *iterative* development process, including:

SC2.5:F5. **Partitioning (or scoping [42]) of WPS produced in the SI phase**

SC2.5:F6. **Partitioning of WPS produced in the SS phase**

SC2.5:F7. **Partitioning of WPS produced in the SR phase**

SC2.5:F8. **Specification of concrete development plans for individual iterations**

SC2.5:F9. **Specification of concrete development plans for inter-phase iterations**

ACKNOWLEDGMENT

This work is funded by the ARC (Australian Research Council), under Discovery scheme no. DP0988345.

REFERENCES

[1] A. Arsanjani, *et al.*, "SOMA: A method for developing service-oriented solutions," *IBM Systems Journal,* vol. 47 (3), pp. 377-396, 2008.

[2] M. Bell, *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2008.

[3] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *2nd International Conference on Software Engineering*, San Francisco, USA, 1976, pp. 592-605.

[4] R. Chapman, "Correctness by construction: a manifesto for high integrity software," presented at the 10th Australian workshop on Safety critical systems and software - Volume 55, Sydney, Australia, 2006.

[5] D. Chappell, *Enterprise Service Bus*: O'Reilly, 2004.

[6] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Transactions on Software Engineering,* vol. 24 (8), pp. 629-639, 1998.

[7] A. Dahanayake, H. Sol, and Z. Stojanovic, "Methodology Evaluation Framework for Component-Based System Development," *Journal of Database Management,* vol. 14 (1), pp. 1-26, 2003.

[8] T. Erl, *SOA: Principles of Service Design*. Indiana, USA: Prentice Hall PTR, 2007.

[9] T. Erl, *SOA Design Patterns*. Indiana, USA: Prentice Hall PTR, 2009.

[10] R. E. Fairley, *Managing and Leading Software Projects*. Hoboken, NJ, USA: Wiley-IEEE Computer Society Press, 2009.

[11] N. Fenton, *Software Metrics: A Rigorous Approach*. London, UK: Chapman & Hall, 1991.

[12] D. Firesmith and B. Henderson-Sellers, *The OPEN Process Framework*. Boston, MA: Addison-Wesley, 2002.

[13] K. Fung and G. C. Low, "Methodology evaluation framework for dynamic evolution in composition-based distributed applications," *Journal of Systems and Software,* vol. 82 (12), pp. 1950-1965, 2009.

[14] F. Garcia, *et al.*, "FMESP: Framework for the modeling and evaluation of software processes," *Journal of Systems Architecture,* vol. 52 (11), pp. 627-639, 2006.

[15] J. Gero and U. Kannengiesser, "A function–behavior–structure ontology of processes," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing,* vol. 21 (4), pp. 379-391, 2007.

[16] A. Gillies, *Software Quality*. London: Chapman & Hall, 1992.

[17] P. Green and M. Rosemann, "Integrated Process Modeling: An Ontological Evaluation," *Information Systems,* vol. 25 (2), pp. 73-87, 2000.

[18] Q. Gu and P. Lago, "Guiding the selection of service-oriented software engineering methodologies," *Service Oriented Computing and Applications (SOCA),* vol. 5 (4), pp. 203-223, 2011.

[19] ISO/IEC, "ISO/IEC TR 9126-1:2001 Software Engineering: Product quality - Quality model," International Organization for Standardization / International Electrotechnical Commission, Geneva2001.

[20] ISO/IEC, "ISO/IEC TR 9126-3:2003 Software Engineering: Product quality - Internal metrics," International Organization for Standardization / International Electrotechnical Commission, Geneva2003.

[21] ISO/IEC, "ISO/IEC TR 9126-2:2003 Software Engineering: Product quality - External metrics," International Organization for Standardization / International Electrotechnical Commission, Geneva2003.

[22] ISO/IEC, "ISO/IEC 25000:2005 Software Engineering: Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE," International Organization for Standardization / International Electrotechnical Commission, Geneva2005.

[23] ISO/IEC, "ISO/IEC 12207:2008 Systems and Software Engineering - Software life cycle processes," International Organization for Standardization / International Electrotechnical Commission, Geneva2008.

[24] N. Jayaratna, *Understanding and Evaluating Methodologies: Nimsad, a Systematic Framework*: The Mcgraw-Hill Information Systems, 1994.

[25] K. Jensen and L. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*: Springer, 2009.

[26] U. Kannengiesser and L. Zhu, "An ontologically-based evaluation of software design methods," *The Knowledge Engineering Review,* vol. 24 (1), pp. 41-58, 2009.

[27] B. Kitchenham, S. Linkman, and D. Law, "DESMET: A Methodology for Evaluating Software Engineering Methods and Tools," *Computing and Control Engineering Journal,* vol. 8 (3), pp. 120-126, 1997.

[28] P. Kroll and P. Kruchten, *The Rational Unified Process made easy*. Reading, USA: Addison-Wesley, 2003.

[29] P. Kruchten, *The Rational Unified Process: an introduction*, 3 ed. Boston, MA: Addison-Wesley, 2003.

[30] P. Kruchten, "Casting Software Design in the Function-Behavior-Structure Framework," *IEEE Software,* vol. 22 (2), pp. 52-58, 2005.

[31] M. Lehmann, "Deploying large-scale interoperable Web Services infrastructures," *Web Services Journal,* vol. 5 (1), pp. 10-15, 2005.

[32] F. Liu, *et al.*, "SaaS Integration for Software Cloud," presented at the IEEE 3rd International Conference on Cloud Computing (CLOUD 2010), FL, USA, 2010.

[33] J. h. Liu, *et al.*, "The impact of software process standardization on software flexibility and project management performance: Control theory perspective," *Information and Software Technology,* vol. 50 (9-10), pp. 889-896, 2008.

[34] M. Mancioppi, *et al.*, "Towards a Quality Model for Choreography," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops. LNCS*. vol. 6275/2010, A. Dan, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2010, pp. 435-444.

[35] T. McBride, "The mechanisms of project management of software development," *Journal of Systems and Software,* vol. 81 (12), pp. 2386-2395, 2008.

[36] P. Mohagheghi, "Evaluating Software Development methodologies based on their practices and promises," in *New Trends in Software Methodologies, Tools and Techniques*, H. Fujita and I. Zualkernan, Eds., ed: IOS Press, 2008.

[37] J. Nielsen, *Usability Engineering*. San Diego: Academic Press, 1994.

[38] D. Norman, *The Design of Everyday Things*: Doubleday Business, 1990.

[39] L. O'Brien, "A Framework for Scope, Cost and Effort Estimation for Service Oriented Architecture (SOA) Projects," in *Australian Software Engineering Conference (ASWEC)*, Gold Coast, Australia, 2009.

[40] OASIS, "Web Services Business Process Execution Language Version 2.0 - Standard," Organization for the Advancement of Structured Information Standards (OASIS)2007.

[41] OMG, "The Software and Systems Process Engineering Meta-Model 2.0 (SPEM 2.0)," Object Management Group (OMG)2008.
[42] L. Padgham and M. Perepletchikov, "Prioritisation mechanisms to support incremental development of agent systems," *International Journal of Agent-Oriented Software Engineering (IJAOSE),* vol. 1 (3/4), pp. 477-497, 2007.
[43] M. Papazoglou and W.-J. van den Heuvel, "Service-Oriented Design and Development Methodology," *International Journal of Web Engineering and Technology (IJWET),* vol. 2 (31), pp. 412-442, 2006.
[44] M. Papazoglou*, et al.*, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer,* vol. 40 (11), pp. 38-45, 2007.
[45] M. Papazoglou and W.-J. van den Heuvel, "Business process development life cycle methodology," *Communications of the ACM,* vol. 50 (10), pp. 79-85, 2007. http://doi.acm.org/10.1145/1290958.1290966.
[46] M. Perepletchikov*, et al.*, "Formalising Service-Oriented Design," *Journal of Software (JSW),* vol. 3 (2), pp. 1-14, 2008.
[47] M. Perepletchikov, C. Ryan, and Z. Tari, "The Impact of Service Cohesion on the Analysability of Service-Oriented Software," *IEEE Transactions on Services Computing,* vol. 3 (2), pp. 89-103, 2010.
[48] M. Perepletchikov and C. Ryan, "A Controlled Experiment for Evaluating the Impact of Coupling on the Maintainability of Service-Oriented Software," *IEEE Transactions on Software Engineering,* vol. 37 (4), pp. 449-465, 2011.
[49] G. Poels and G. Dedene, "Distance-based software measurement: necessary and sufficient properties for software measures," *Information and Software Technology,* vol. 42 (1), pp. 35-46, 2000.
[50] J. Recker*, et al.*, "Measuring Method Complexity: UML versus BPMN," in *Proceedings of the 15th Americas Conference on Information Systems*, San Francisco, CA, USA, 2009.
[51] M. Rossi and S. Brinkkemper, "Complexity Metrics for Systems Development Methods and Techniques," *Information Systems,* vol. 21 (2), pp. 209-227, 1996.
[52] R. Sharble and S. Cohen, "The object-oriented brewery: a comparison of two object-oriented development methods," *ACM SIGSOFT Softw. Eng. Notes,* vol. 18 (2), pp. 60-73, 1993.
[53] K. Siau and R. Matti, "Evaluation of Information Modeling Methods -- A Review," in *The Thirty-First Annual Hawaii International Conference on System Sciences*, 1998.
[54] K. Siau and Q. Cao, "Unified Modeling Language: A Complexity Analysis," *Journal of Database Management (JDM),* vol. 12(1) (1), pp. 26-34, 2001. doi:10.4018/jdm.2001010103.
[55] H. Sol, "Information systems development: a problem-solving approach," in *Challenges and strategies for research in systems development*, ed: John Wiley & Sons, Inc., 1992, pp. 151-161.
[56] X. Song and L. Osterweil, "Toward Objective, Systematic Design-Method Comparisons," *IEEE Software,* vol. 9 (3), pp. 43-53, 1992. http://dx.doi.org/10.1109/52.136166.
[57] S. Stevens, "On the Theory of Scales of Measurement," *Science,* vol. 103 (2684), pp. 677–680, 1946.
[58] Y. Wand and R. Weber, "An Ontological Model of an Information System," *IEEE Transactions on Software Engineering,* vol. 16 (11), pp. 1282-1292, 1990. 10.1109/32.60316.
[59] O. Zimmermann, M. Tomlinson, and S. Peuser, *Perspectives on Web Services: Applying SOAP, WSDL, and UDDI to Real-World Projects*: Springer Professional Computing, 2003.

Dr Mikhail Perepletchikov received B.App.Sc. (Computer Science) Honours 1st Class degree from RMIT University, Melbourne, Australia in 2004 and completed his PhD (Computer Science) at the same institution in 2009. His PhD thesis addressed software quality in the context of SOA. He is currently a Research Fellow and Sessional Lecturer in the School of Computer Science and Information Technology, RMIT University. He is a member of the IEEE, the IEEE Computer Science Society, and the Australian Computer Society (ACS).

Dr Caspar Ryan completed a B.App.Sci. Comp. Sci. (Hons) in 1996 and received his PhD in Computer Science, from RMIT University Melbourne Australia in 2002. His PhD thesis title was A Methodology for the Empirical Study of Object-Oriented designers in which he presented a novel approach for studying software engineers in practice. He is currently a Senior Lecturer in the School of CS & IT at the same institution. His current software engineering research involves metrics and software development methodology for SOA as a joint CI on an Australian Research Council Discovery grant.

Prof Zahir Tari is a full professor at RMIT University and Director of the DSN (Distributed Systems & Networking) discipline at the School of Computer Science & IT, RMIT (Australia). He graduated with an honours degree in Operational Research at USTHB (Universite Houari Boumediene) in Algiers (Algeria, 1984), Master degree in Operational Research at University of Grenoble (France, 1985), and a PhD degree in Computer Science at University of Grenoble (France, 1989). His recent primary research interests are in system's performance (such as load balancing under various traffic conditions) and security (such as access control and information flow control). More details about Professor Tari and his team can be found at http://www.cs.rmit.edu.au/dsn.