# Review of Relevant System Development Life Cycles (SDLCs) in Service-Oriented Software Engineering (SoSE)

L. Rodríguez-Martínez*[1], M. Mora[2], F. Álvarez[3], L. Garza[4], H. Durán[5], J. Muñoz[6]

[1] Technological Institute of Aguascalientes
Adolfo López Mateos Avenue, 1801
Aguascalientes, Aguascalientes, México
[2,3,4,6] Autonomous University of Aguascalientes
Universidad Avenue, 940
Aguascalientes, Aguascalientes, México
[5] Universidad de Guadalajara
Periférico Norte, 799
Zapopan, Jalisco, México
*lrodriguez@mail.ita.mx

**ABSTRACT**
Service-oriented software engineering (SoSE) is a new paradigm for building software systems, fostered by the availability of a new -but already mature- computing technology based on services. SoSE advances the current object-oriented and the component-based software engineering paradigms. Under that new paradigm, multiple software-system development life cycle (SDLC) methodologies have been proposed; however, none of them have gained a total acceptance as the dominant SDLC in SoSE. On this theoretical and practical situation, we believe that a research is required to reach more standardized and stabilized knowledge about SDLCs in SoSE. Thus, this article reviews nine recent SDLCs proposed for SoSE with the aim to present a descriptive-comparative landscape of a relevant range of SDLCs for SoSE. Such description-comparison is guided by two criteria: (i) the extent of completeness of each SDLC, with respect to the proposed phases, activities and delivered artifacts, and (ii) the extent of the Boehm-Turner's Rigor-Agility balance. Our results suggest that only three of the nine SDLCs studied already provide the best level of completeness and Rigor-Agility. Finally, we consider that the reported descriptive-comparative framework and their findings from each SDLC can be useful also for comparing and elaborating future SDLCs in SoSE.

Keywords: Service-oriented software systems (SoSS), service-oriented software engineering (SoSE), system development life cycle (SDLC), software development methodologies, Boehm-Turner's Rigor-Agility balance level.

**RESUMEN**
La ingeniería de *software* orientada a servicios (SoSE – *service-oriented software engineering*) es un nuevo paradigma para construir sistemas de *software* que ha florecido por la disponibilidad de una nueva pero madura tecnología computacional basada en servicios. SoSE es un avance sobre los paradigmas de la ingeniería de software orientada a objetos (OOSE – *object-oriented software engineering*) y basado en componentes (CBSE – *component-based software engineering*). En SoSE, se han propuesto múltiples metodologías de ciclo de vida de desarrollo de sistemas de software (SDLC – *software-system development life cycle*), sin que alguna de ellas sea aceptada como SDLC dominante en SoSE. Bajo esta situación, se sugiere investigar con el objetivo de un conocimiento más estandarizado y estabilizado sobre SDLC en SoSE. Este artículo revisa nueve SDLC para SoSE recientes, presentando un panorama descriptivo-comparativo de un rango relevante de ellos. Esta descripción-comparación se guía por dos criterios: (i) el grado de completitud de cada SDLC respecto a sus fases, actividades y artefactos entregables, y (ii) el grado de balance *rigor-agility* (Boehm-Turner). Los resultados sugieren que tres de los SDLC bajo estudio son los más completos y balanceados. Finalmente, se sugiere la utilidad del *framework* de descripción-comparación y los hallazgos reportados para la elaboración de algún futuro SDLC para SoSE.

## 1. Introduction

Service-oriented software engineering (SoSE) is a new paradigm of software engineering which is focused on the design and implementation of service-oriented software systems (SoSS) [1]. SoSE can be defined as (i) the application of a quantifiable and disciplined approach for the

development, operation and maintenance of SoSS; and (ii) the study of approaches referring to point 1 of this definition. SoSE advances the current Object-oriented [2] and Component-based Software Engineering [3] paradigms (OOSE, CBSE), based on the availability of an already mature computing technology for building software services.

While the concepts of class/object and component are the fundamental entities in OOSE and CBSE, the concept of service appears in SoSE as the key design entity. A service can be defined -in general terms- as a fully implemented and expected functionality provided from an entity (service provider) to other entities (service customer or customers).

Under this new SoSE paradigm, as in the current OOSE and CBSE paradigms, the construction and maintenance of software systems demands a software-system development life cycle (SDLC). A SDLC covers the entire spectrum of required activities -from system conception to system disposal- for building a software system. A SDLC is usually realized by a development methodology (e.g., phases, activities, roles, artifacts, and tools) that covers the conception, building, and deployment of a software product (i.e. the software system). In the software engineering literature, SDLCs (and their realization methodologies) have been proposed since 1970. According to Rodriguez et al. [4], their evolution or adaptation has occurred because of two main forces: (i) a knowledge-gap driver, which is manifested in a process engineering weakness detected in the utilization of models for new software developments cases, and/or (ii) a technological change driver, which is manifested in the radical improvement or introduction of a new computing technology. Additionally, important researchers in the software engineering stream [5, 6] have suggested the extent of Rigor-Agility of a SDLC as other driver force that fosters the evolution or adaptation of a SDLC. In particular, Bohem [7] suggested the need for rigor in very large-scale projects, but Boehm and Turner [8] also recommend that for most usual projects a balance between rigor and agility levels is required in modern SDLCs. As they indicate [8, pp. 1]: "*every successful venture in a changing world requires both agility and discipline. If one has strong discipline without agility, the result is inflexible hierarchy and stagnation. Agility without discipline leads to the heady, unencumbered enthusiasm of a start-up company–before it has to turn a profit. … Great companies, and great software projects, have both in measures appropriate to their goals and environment*."

In spite of such accomplishments in SoSE, we have identified a research need for more standardized and stabilized knowledge about SDLCs in SoSE; this is due to the fact that while several SLDCs have been proposed in SoSE, none has gained the majority acceptance in academic and/or practitioner settings. Consequently, academics and practitioners have to use ad-hoc adaptations of SDLCs designed for previous paradigms (e.g. OOSE or CBSE) instead of more adequate SDLCs (e.g. designed specially in the SoSE paradigm).

Thus, in this article, we review nine recent SDLC proposed for SoSE with the aim to present a descriptive-comparative landscape of plausible SDLCs in this new paradigm. Such description-comparison is guided by two criteria: (i) the extent of completeness of the proposed phases, activities and delivered artifacts, and (ii) the extent of the Boehm-Turner's Rigor-Agility balance. The remainder of this paper continues as follows: in Section 2, theoretical fundaments for this research are reported (e.g. 2.1. is a conceptual review of the evolution of SDLCs, 2.2 is a review of fundamental concepts in SoSE, 2.3 is a review of MDA principles, 2.4 is a review of Rigor-Agility concepts, and 2.4 is the SDLC framework for description and comparison). In Section 3, a systematic description and comparison of nine SDLCs reported recently in SoSE stream is reported. Finally, in Section 4, conclusions, limitations and recommendations to advance this research are reported.

## 2. Conceptual Foundations on Service-Oriented Software Systems (SoSS)

### *2.1 Evolution of SDLCs*

In Rodriguez et al. [4], thirteen SDLCs from the software engineering literature (traditional SDLCs) were reviewed to elaborate a state-of-the-art map on SDLC evolution. As a result of such study, it was determined that SDLCs in software engineering are not service-oriented SDLCs [4]. To develop an SDLC evolution map, three concepts were used: (i) methodological era, (ii)

rigor level of SDLCs, (iii) and agility level of SDLCs. Methodological era is posed by Avison & Fitzgerald's [9], and provides a chronological four-period evolution-classification scheme (e.g., pre-methodology era, early methodology era, methodology era, and post-methodology era). Rigor level of a SDLC represents the extent of detailed specifications to be achieved in each phase, activity and task before proceeding to the next phase. Agility level of an SDLC accounts for the extent of a system developer's freedom to complete phases, activities and tasks in the application of the SDLC. These two concepts were assessed by using ordinal scales of four and two values, respectively (for rigor: null, low, medium, and high; and for agility: low and high).

A non-trivial inference on the utilization of these scales is the assertion that both concepts (rigor and agility) do not represent disjoint sets [10]: i.e., the notion of an intersection in both SDLC rigor and agility sets of scales is supported. In this way, an SDLC can be balanced when it has either medium or high assessment in both rigor and agility levels. It is also possible to name a partially-balanced SDLC when the assessments are high and medium or vice versa for both rigor and agility levels. Hence, in this research both concepts are not considered antonymous.

In Figure 1, the SLDC evolution map is presented. For each SDLC, the following items are reported: (i) year of origin, (ii) its main change force (a methodological knowledge gap or the emergence of a new technology), and (ii) its extent of rigor and agility [4, 8]. In Figure 1, symbols "●" and "◒" suggest the suitable locations for balanced and partially balanced SDLCs, respectively. Under such perspective, a { (no rigor, no agility) → (rigor, no agility) → (no rigor, agility) } evolution can be identified. In particular, Rodriguez et al. [4] suggest that in software engineering principles and foundations, rigor specification takes precedence over agile attributes. As evidence, two of the three eras start with at least a medium rigor level, and each new SDLC usually evolves from a previous SDLC – e.g., it re-uses most content than previous ones. However, generic industrial pressure, for jointly reducing manufacturing time cycles and keeping high-quality products and services, also suggests the need for a trade-off between rigor and agility specifications in future SDLCs. Hence, a next generation of SDLCs might be predicted when a new development technology emerges and/or new critical knowledge process engineering gaps are identified.

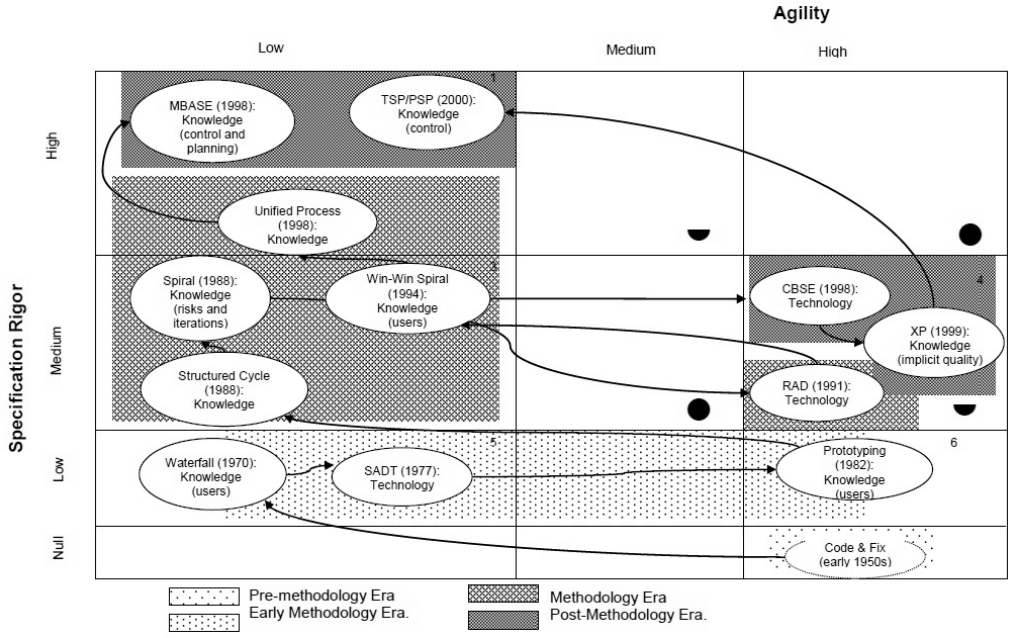


Figure 1. Evolution map of SDLCs with possible location for rigor-agility SDLCs.

Currently, the occurrence of two events can suggest the need for a new SDLC approach: (i) new computational and information technology development tools oriented to services are available [11, 12, 13], and (ii) existence of critical knowledge gaps about service-oriented software engineering (SoSE) [14, 1]. Furthermore, in such emergent SoSE paradigm, some studies [15, 11, 16] suggest a greater complex development process than in previous SDLCs of past eras, because a service business layer is added to the usual system and computational layers. Consequently, research on current progress on SDLCs in SoSE can contribute to the advance of the theoretical and best practices knowledge on foundations and engineering issues.

*2.2 Key concepts on service-oriented software systems*

*2.2.1 Service concept*

According to Dumas [17], the service concept comes from the business domain when an enterprise, group or a single person (called the service provider) develops one or a set of activities for clients (called service customers). In general, the channels for providing these services can be people or computer-based systems. A service can be differentiated also as "manual" (from person to person), or "automated" (from a computer to another computer or person). In the context of SoSE, a service can be also interpreted in two senses: (i) a ***business service*** (when we do not specify the mean or channel to provide it), and (2) a ***computing service*** (when an IT is the mandatory mean or channel for providing it).

Dumas [17] further classifies ***computing services*** into two types: (a) ***computing service of business*** (when it implements a ***business service***), and (b) ***ICT computing service*** which is a ***computing service*** (when it provides a service to another software system). This division between ***computing service*** and ***business service*** -on the consideration that a ***computing service*** implements a business service-, implies that a ***computing service*** can implement services for a user external utilization (i.e. a ***business service***) and/or for internal utilization (i.e., another ***computing service*** as its internal customer). Table 1 summarizes these concepts.

A complementary conceptualization of services –based on business service literature- [19] is toward the need for value appreciations that are mutually and explicitly agreed –through a contract – by providers and customers. Under such perspective, and Dumas´s service interpretation, in this paper service is defined as "***an intangible functionality***

| Type of service | Division | Definition | Mean of a service composition |
|---|---|---|---|
| Business Service | Traditional business service in administration | A business service that is provided by a person or enterprise directly to a client. | A business service composition constitutes a work-flow in an enterprise, but which is perceived a functional whole. |
| Computing Service | Business computing services | A computing service that is implemented to provide system's users with a business service. | A composition of business computing services can constitute a task, a business process, a work-flow, or even a service-oriented software system (SoSS). |
| | ICT computing services | A computing service which provides the software system with a service and it is based on additional ICT services. | An ICT computing service can be provided by a unique service, or by a service-computing composition. |

Table 1. Service concept view and the meaning of Service Composition [18].

***(which uses also tangibles goods) that a provider enables for customers to benefit through a channel, and where its value and cost is mutually agreed in an implicit or explicit contract, and determined in terms of several properties –defined also mutually by customer and provider- (e.g., security, quality, and reputation, among other properties)***"

*2.2.2 SoSS concept*

A service-oriented software system (SoSS) refers to a distributed and loosely-coupled software system which is constructed based on the definition and implementation of a suite of services that forms it [13]. A typical SoSS owns the following attributes (Table 2 is extended and adapted from [20] and [21]):

In particular, web services can be considered as the most deployed technology for SoSS. However, we consider that key service concept for SoSE not necessary must be deployed using such technology. Thus, in this study services are not synonymous of web services.

*2.2.3 MDA-based SDLC SoSE framework components*

Rodríguez et al. [22] developed an initial descriptive and comparative SoSE SDLC framework which is derived from (a) a model-driven architecture (MDA) framework [23], (b) a service-oriented analysis and design (SOAD) approach [15], (c) a set of software service conceptualizations [11], and (d) a three-phased macro model [24]. Table 3 reports a summarized scheme of the first three conceptual schemes for SDLCs in SoSE.

The MDA is an abstract system development process where none methodological detail on how to generate its deliveries is reported [23]. MDA is based on the design paradigm where the definition and implementation of system architecture must be independent of its realization technology. The MDA abstract process defines three deliveries (models) based on three related viewpoints: (1) computing independent model (CIM) is focused on the system requirements and its environment without any

| Attribute | Object-oriented Software System | Component-based Software System | Service-oriented Software Systems |
|---|---|---|---|
| Key analysis entity | Class | Business component | Business service |
| Key design entity | Object (conceptual) | Component (conceptual) | Business computing service |
| Key building entity | Object (local runtime) | Component (local or distributed runtime) | ICT computing service |
| Coupling level with remainder software | Tightly | Medium | Loosely |
| Cohesion level | Normal | High | Very high |
| Platform Interoperability | Minimal or null | High | Very high |
| Typical technology | C++ | JavaBeans | Web services from several languages (Java, C#, PHP) |

Table 2. Comparative Table of OOSE, CBSE y SOSE paradigms.

| MDA [23] | SOAD Layers [15] | Service-oriented Products [11] |
|---|---|---|
| CIM – Computation independent model | Business | Service identification |
| PIM – Platform independent model | Architecture | Service specification<br>Service realization<br>Service orchestration (service composites)<br>Service choreography |
| PSM – Platform specific model (for its generation a platform model is needed) | Application | Service implementation specification |
| Platform executable model | | Service implementation. |

Table 3. SDLCs SOSE Fundamental Schemes.

technical consideration of computing platform, (2) platform independent model (PIM) is focused on detailed system operational specifications but not instanced for a specific platform, and (3) platform specific model (focused on the specific implantation). From its official specification [23], a fourth delivery can be inferred. In this study, this fourth model is called the *platform executable model* (PEM) focused on the final runtime model.

The service-oriented analysis and design (SOAD) [15] approach attempts to address three domains (business, architecture and application) or levels of abstraction through the integration of business model process (BMP), enterprise architecture (EA) and object-oriented analysis and design (OOAD) approaches in two phases: analysis and design. Our adaptation of the SOAD approach considers the complete SDLC by aggregating the development phase. Such adaptation is presented in Figure 2.

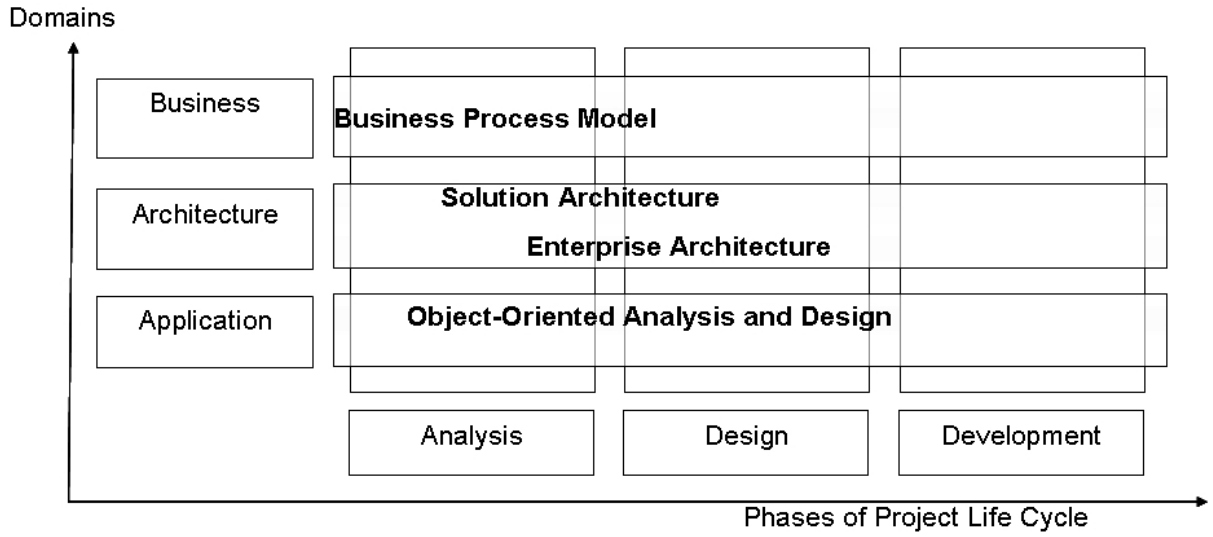


Figure 2. SOAD scheme [adapted from Zimmermann [15].

SOAD is elaborated using two dimensions: (1) the vertical dimension represents the three addressing domains, and (2), the horizontal dimension represents the phases of an SDLC. Four generic activities are proposed in SOAD: ***business process model***, ***enterprise architecture***, ***solution architecture***, and ***object-oriented analysis and design***. These activities can be performed (in a separate way) with the BPM, EA y OOAD approaches or in an integrated way with SOAD.

In Rodríguez et al. [24], a similar scheme -based on systems engineering- with three macro-phases is reported. Such macro-phases are (1) system definition, (2) system development, and (3) system evolution. From these previous schemes, Rodríguez et al. [22] proposed an integrated framework, as shown in Figure 3.

Figure 3 presents the MDA-based SDLC for SoSS which is used as the conceptual base for the descriptive-comparative framework for this study. This MDA-based SDLC for SoSS is presented as an approach extended and adapted from SOAD in two dimensions: (1) levels of abstraction (business, architecture, application) and, (2) generic phases of SDLC defined in Rodriguez et al [24] (requirements, design, construction and operation, which are grouped in macro-phases definition, development and evolution). Nine generic activities (A1 .. A9) are proposed. Also, each level of abstraction is matched with each MDA model: Business level with computation independent model (CIM), architecture level with platform independent model (PIM), and application level with platform specific model (PSM). Additionally, application level is also matched with

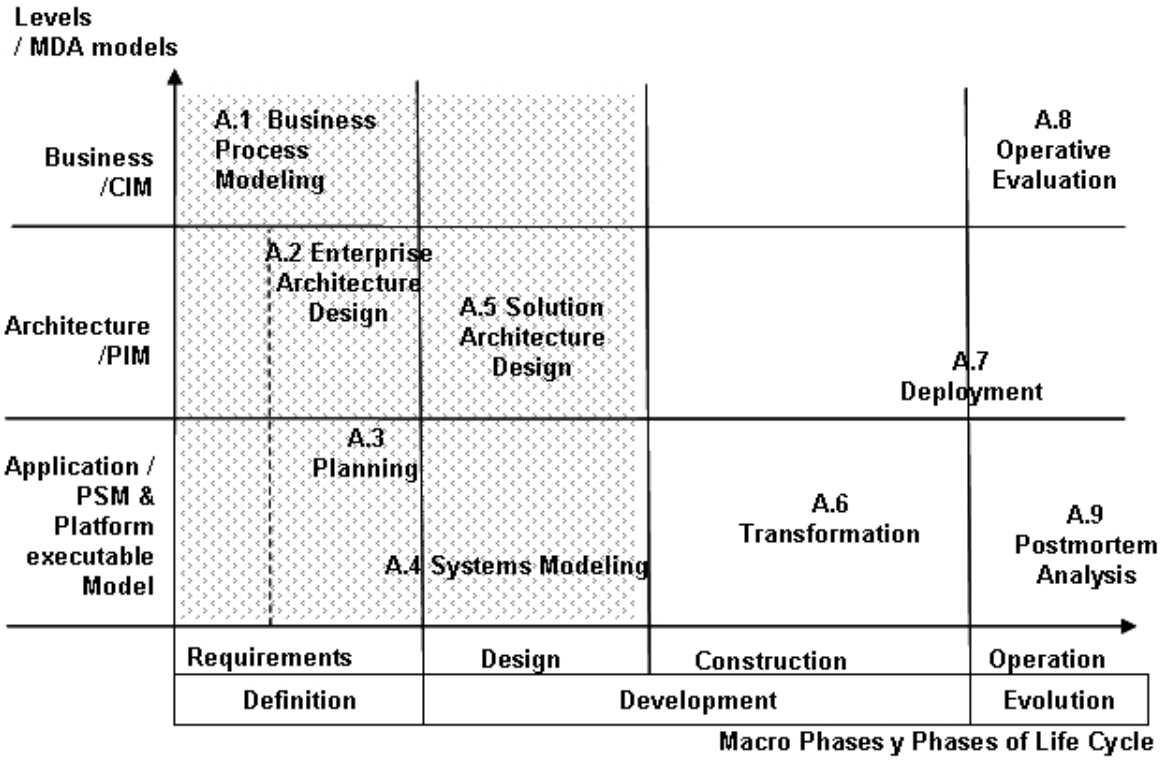


Figure 3. The generic MDA-based SDLC for SOSS.

platform executable model proposed in [22] for the executable model of the system implemented on a specific computing platform.

*2.3 The Boehm-Turner's rigor-agility scheme*

Figure 4 shows the rigor-agility SLDC assessment scheme [8] and its assessment scale (data are from M2 methodology, which is one of the final nine SoSE SDLCs analyzed in this study). The scale uses the values from 0 to 4 as follows: 0 means that the issue is not considered, 1 means the issue is a little considered, 2 means the issue is considered to some extent, 3 means the issue is amply considered. This assessment scheme uses three main factors. **Factor 1** (levels of concerns of the SDLC) assesses the organizational scope for which the SDLC provides specific guidance (i.e., how much the SDLC addresses the expected assumptions and where fewer concerns imply greater agility). Factor 2 (system development life cycle coverage) assesses the life cycle activities that the SDLC have (i.e., how much the SDLC covers the expected phases and activities). In this case, less coverage implies a more agile SDLC. Factor 3 (sources of constraints in the SDLC) assesses the extent of mandatory restrictions to be respected by the developers. Fewer restrictions imply a more agile SDLC. The source of constraints also indicates what risks are covered.

*2.4 The descriptive-comparative framework for SoSE SDLCs*

Based on the previous theoretical and practical MDA, SoSE and rigor-agility concepts, we propose a descriptive-comparative framework as presented in Table 4. This framework is proposed in two dimensions: (1) the phases of the MDA-based SDLC, and (2) the required domain levels (business, architecture and application) to consider a SDLC as a complete one.

| | Factor 1 | | | | | Factor 2 | | | | | Factor 3 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Level of Concerns** (+ rigorous = - assumptions) | | | | | **Life-Cycle Coverage** (+ rigorous = + coverages) | | | | | **Sources of Constraints** (+ rigorous = + restrictions) | | | | |
| | Agility ← + ASSUMPTIONS − → Rigor | | | | | Agility ← − COVERAGE + → Rigor | | | | | Agility ← − RESTRICTIONS + → Rigor | | | | |
| **SDLC FOR SOSS** | **Business Enterprise** | **Business System** | **Multi-team project** | **Single-team project** | **Individual** | **Concept Development** | **Requirements** | **Design** | **Development** | **Maintenance** | **Management Processes** | **Technical Practices** | **Risk / Opportunity** | **Measurement Practices** | **Customer Interface** |
| **M2** | **1** | **1** | **0** | **0** | **0** | **0** | **1** | **1** | **1** | **0** | **0** | **2** | **1** | **0** | **1** |

Figure 4. The Boehm-Turner's rigor-agility SDLC assessment scheme.

| "Definition" Macro-Phase: | Phase | (Generic Expected) Artifacts | Domain | Activities of MDA-based SDLC | Specific SDLC's activities |
|---|---|---|---|---|---|
| | ***Requirements*** | CIM | Business | **Business Process Modeling (BMP)** | |
| | | | Architecture | **Enterprise Architecture Design** | |
| | | | Application | **Planning** | |
| **"Development" Macro-Phase:** | **Phase** | **(Generic Expected) Artifacts** | **Domain** | **Activities of MDA-based SDLC** | **Specific SDLC's activities** |
| | ***Design*** | PIM , PSM | Business | **None proposed activity** | |
| | | | Architecture | **Solution Architecture Design** | |
| | | | Application | **System Modeling**<br>**(Composition, Coordination and Orchestration)** | |
| | ***Construction*** | Platform Executable Model | Business | **None proposed activity** | |
| | | | Architecture | **None proposed activity** | |
| | | | Application | **Transformation**<br>**Deployment** | |
| **"Evolution" Macro-Phase:** | **Phase** | **(Generic Expected) Artifacts** | **Domain** | **Activities of MDA-based SDLC** | **Specific SDLC's activities** |
| | ***Operation*** | E&EP | Business | **Operative Evaluation** | |
| | | | Architecture | **Deployment** | |
| | | | Application | **Deployment**<br>**Postmortem Analysis** | |

Table 4. Descriptive-comparative SoSE SDLC framework.

The framework represents each possible combination between the three levels (business, architecture, and application) and four phases (requirements, design, construction, and operation) of the MDA-based SDLC for SOSS. Thus such generic activities are classified through the list level-phase over considering each generic activity would be part of different levels, and at the same time, we keep the generic MDA models (CIM, PIM, PSM) separated by the reclassification of the generic activities into CIM, PIM, PSM, platform executable model (executable PM) and evaluation and evolution plan (E & EP). In this reclassification, it is proposed that CIM is the generic expected artifact of the requirements phase, PIM and PSM are the generic expected artifacts of the design phase, executable PM is the expected artifact of the construction phase and E & EP are the expected artifacts of the operation phase. The framework enables a uniform comparison of SDLCs and helps to find methodological gaps in emerging SoSE SDLCs.

## 3. Application of the descriptive-comparative framework for SoSE SDLCs

Using the descriptive-comparative framework for SoSE SDLCs reported in Table 4, we review nine relevant SDLC methodologies identified in the core SoSE literature during the 2000-2007 period (see Table 5).

For this aim, firstly each SDLC is described in a synthesized mode. Secondly, the activities proposed in each SDLC are placed in the most suitable position of the descriptive-comparative SoSE SDLC framework. Using this particular criterion, all activities proposed in each SoSE SDLC can be compared more uniformly given that they are compared versus a unique and similar structure rather than between them. Thirdly, the rigor-agility assessment framework for SDLCs is used to assess each methodology on such criterion.

| ID | SDLC Name | Source |
|---|---|---|
| M1 | WSbBP (Web services-based business processes methodology) | Karastoyanova, 2003 [25] |
| M2 | COMPOSE (COMPonent-oriented software engineering) | Kotonya, 2004 [3] |
| M3 | Toward an SODM (Toward a *service-oriented development methodology*) | Ivanyukovich et al., 2005 [26] |
| M4 | Toward an SODM for the orchestration and validation of cooperative e-business components | Kühne et al., 2005 [27] |
| M5 | SOA-LC management (Service-oriented architecture life cycle management). | Cox et al., 2005 [28] |
| M6 | Automatic derivation of BPEL4WS from IDEF0 process models | Karakostas et al., 2006 [29] |
| M7 | BP-DLC (Business process development life cycle) | Papazoglou et al., 2007 [30] |
| M8 | A stakeholder-driven SOA-LC model (SOA-LC – Service-oriented architecture life-cycle model). | Qing and Lago, 2007 [31] |
| M9 | Rational unified process for systems engineering (RUP-SE) | Rational, 2003 [32] |

Table 5. Nine relevant 2000-2007 SoSE SDLCs.

*3.1 Description of the nine relevant 2000-2007 SoSE SDLCs*

This section presents a brief description of each one of the nine SoSE SDLCs. We identify the main characteristics associated with MDA, SoSE, and rigor-agility issues in each SDLC.

**M1.WSbBP: Web service-based business processes methodology [25]**

WSbBP is based on web services technology. It relies on the concept of business work-flows called "web service-based business processes" or "WS-flows" (web service flows). According to authors [25], WS-flows are significant compositions of tasks for business problem solutions which demand the conversion of such tasks in discrete web services, in a predefined order according to business rules rigorously specified. WSbBP does not predefine specific tools and protocols for using in the development of WS-flows, but several items are recommended (XML, BPEL, BPML, XSLT, and BPWS4j). WSbBP proposes six activities: (1) "*process template modeling and assembly*", (2) "*process definition generation*", (3) "*pre-processing*", (4) "*deployment*", (5) *"execution time* (services coordination) " and (6) "*post-run time* (analysis)". WSbBP is focused on the building of business flows which can be implementable in some agreed web services-based technology. It is required to count with several enabled web services already orchestrated on business flows and coordinated by coordination protocols according to a B2B[1] standard. This SDLC intends a fast and easy development of business processes via web services code generation automation. This approach is suggested to reduce the manual work of developers, to help an easy creation of business flows, and to promote code reusing, abstraction (*via templates*), and take advantage of web service flows technology. While this SDLC is strong through its contribution of web services code generation automation, it also lacks of specific process guidance on requirements definition, system architecture design, and system evolution. However, Karastoyanova's study [25] contributes with one of the first generic methodologies to define, build and reach an executable business process based on web services.

[1] B2B – Business to Business. It refers to electronic commerce between enterprises. It is a type of service in which the service provider is a business organization and the consumer is also a business organization.

**M2. COMPOSE: COMPonent-oriented software engineering [3]**

COMPOSE extends the original computing-oriented service concept toward a business service in the requirements definition activity. It provides a framework for mapping requirements to a hybrid component and service-oriented architecture. COMPOSE is a dual business-management and computing service-based approach, which supports also a hybrid component-service oriented development. COMPOSE fosters also a reuse process. The method proposes two great set of activities: planning and development. The planning stage is not detailed in the source paper. In the development stage there are four activities: (1) requirements engineering, (2) design and composition, (3) verification, and (4) negotiation. The authors indicate techniques to use in each phase and address challenges that organizations face up when intend to adopt a "service-oriented development". COMPOSE realizes the need for more adequate SDLCs to develop SoSS and for minimizing the implicated risks. The authors realize also the need to develop systems with hybrid software models, where components and services co-exist in a same system. According to authors, the proposed method incorporates negotiation as a key process to balance aspects of systems requirements and of business constraints, with assumptions and capacities of the architecture that are implicated in the component and services software.

**M3. T-SODM: toward a service-oriented development methodology [26]**

This study does not report a single SDLC per se but a set of design guidelines and four plausible SDLCs (XP, RUP, TROPOS and MAP). It presents also a structured approach to analyze such four SDLCs under specific concerns of service-oriented applications. In this study, the lack of a compressive methodological approach to develop SoSS is also remarked. Key software design concepts for understanding the evolution toward SoSS are also reported as follows: (1) service-oriented, (2) object-oriented and (3) component-oriented. T-SODM proposes the study of existing SDLCs over three dimensions: (1) "*managing change in software development*", (2) "*specifying the software development process*" and (3) "*targeting the stakeholder goals*". For each dimension it identifies SDLCs that are capable of reaching the characteristic challenges SoSS toward definition of a *service-oriented development methodology* (SoSE SDLC). Main findings of each dimension related to existent SDLCs are (a) the use of some characteristics of XP for an agile development, (b) characteristics of RUP to specify the development process, and (c) selection of two goal-oriented approaches for the requirements specification as stakeholders' goals (TROPOS and MAP). Four existent SDLCs (XP, RUP, TROPOS and MAP) are reviewed and authors conclude that all of them contribute to service domain, but adaptations for addressing specific domain characteristic are required. However, none specific proposals are reported for implementing such adaptations. Furthermore, not a new SoSE SDLC is reported.

**M4. T-SODM: toward a service-oriented development methodology [27]**

T-SODM is more focused on the computing layers (PSM or ICT service concepts) than in a balanced SoSE SDLC. Customer needs are roughly described as high level business descriptions which manually derive in implementation models. T-SODM proposes a first view for a SDLC which enables automation of developments steps through combined approaches of requirements analysis, transforming and checking models. This semi-automated development process should enable efficiency and quality improvements.
In this SDLC, a SoSS is developed through the design and semi-automated implementation of orchestrated services compositions. Such last items are realized by transforming the high level definition of business requirements toward a technical specification which is able to be executed. T-SODM proposes system-orchestration of distributed cooperative components by using semi-automation tools to build part of SoSS. In T-SODM three activities are proposed: (1) structured analysis of requirements, (2) validation, and (3) transformation. The last delivered product by this SDLC is an orchestrated model skeleton (i.e., a model based requirements specification) which by manual development is completed into an executable orchestration model. Hence, while the notion of an automated or semi-automated design

is powerful, its implementation is not trivial for integrating complex e-business scenarios. In particular, just three activities of nine plausible are proposed in T-SODM. It can be considered as a weakly completed SDLC.

**M5. SOA-LC management: service-oriented architecture life cycle management [28]**

This study does not report an SDLC per se. However, it makes the case for a service-oriented architecture life cycle management process. According to authors [28], SOA is built on a distributed system but it advances on some of their derived problems (a reliable integration of disparate applications, a seamless reusability, and a demand for a variety expertise on different computing technologies implementations). However, although SOA addresses such limitations, deployments based on services also introduce new managerial and control issues in the development of SoSS. For instance: development and test of applications composed by operational service components, deployment and provision of distributed secure and efficient applications based on services through organizational frontiers, and tracking the impact of the business services on the business process that these services are supporting. Authors [28] pose that the development and deployment of SOA relies on a real word viewpoint, in which a set of services are assembled and reused for adapting to new business needs. This flexibility is pursued by organizations as a core value of SOA trying to reach wide transformations over how software is constructed. In particular, this SDLC proposes two set of activities: (1) Preproduction and (2) Production. It also adds control points in each activity of these sets. Included activities are not explicitly reported.

**M6. BPEL4WS automated approach [29]**

M6 proposes an automated process to derive an executable service composition from a process model specified in IDEF0 format. M6 arguments that in the business-service level, the IDEF0 models enable an effective capture of business processes requirements. Constructs of IDEF0 are used for coding requirements of service web execution (these requirements are input, web service parameters, and outputs). Then, this IDEF0 specification can eventually to be translated to a set of web services, including specifications on how these services interact, and on how the business logic which controls them must be executed. According to authors [29] such IDEF0 specifications capture all required information for service orchestration of web services at run-time. M6 describes a *business process management engine* (CLSM Prototype Engine) which analyzes a XML definition of an IDEF0 model, identifies how web services interacts, and automatically generates orchestration code in the selected orchestration language. The authors of this related work create a module that enables to a "CLMS Prototype Engine" for generating executable process descriptions into BPEL4WS. These process descriptions can eventually be executed in a BPEL motor (e.g. "Oracle's BPEL Process Manager"). This approach permits a top-down analysis for business process and their web services. This also avoids inconsistency things between the web service process and its corresponding business process toward maintenance of architectural integrity. Hence, the specification of complex scenarios is limited to the expressivity power of the IDEF0 diagrams and the translator of such diagrams to the execution code.

**M7. BP-DLC: business process development life cycle [30]**

In this study is strongly defended the need for counting on a well-defined SDLC for efficiently and effectively designing, building, monitoring and managing the expected SoSS enterprise applications which must support the set of usually agile and complex business process. Thus, the authors propose design guides to ensure the autonomy and self-containing of services and achieving modular business processes with clear frontiers defined and with discrete end-point services. Such service must be of low-coupling and high cohesion. M7 is a well-defined SDLC (from a more traditional viewpoint than a service-oriented view) with six activities: (1) planning, (2) service and process analysis and design, (3) construction and testing, (4) provisioning, (5) deployment, (6) execution and monitoring. Its definition of BP-DLC is also the most complete SDLC because in an explicit/implicit way it proposes for each of the six phases: its description, a few sets of activities, deliverables

and tools. Also BP-DLC includes some emergent activities such as service requirements, business modeling, service architecture design, and service orchestration. Transition through these activities tends to be incremental and iterative by nature and can imply reviews in situations where their scope cannot be defined totally a priori.

**M8. SOA-LC model: A stakeholder-driven service-oriented life cycle model [31]**

This study also claims for new SDLCs for building SoSS. As authors [31] report: "… *because new roles and new development tasks are introduced in service-oriented development as opposed to traditional software engineering, a new approach to service life cycle management is required*…". The SOA-LC model defines an SOA stakeholder as any entity that covers computing architectural roles including service providers, service consumers (or application providers) and service brokers. In this study, it is observed that current proposals lack clear indications on how SOA stakeholders and service life cycle stages (design time, run time and change time) interact. M8 proposes that SoSS can be developed in two parts like two applications: (i) a service provider that provides the services to their use, and (ii) a service consumer that is composed of several workflows, service compositions and user interfaces that access the services published by the service provider (as end-user applicatin). Hence, a SoSS is a system comprising the service provider, the broker and the consumer. SOA-LC does not define phases, only proposes particular SDLC activities for the "SOA stakeholders": (1) for "service provider", activities are *market scan, requirements engineering, business modeling, service design, service development, service testing, service publishing, service provision, service monitoring, service management*; (2) for "service broker", the activities are *registry selection, registry update* and *registry maintenance*; and finally, (3) for "service consumer", the activities are *requirements engineering, application design, implementation and module testing, service orchestration/composition, service negotiation, service invocation, application testing, service monitoring,* and *application maintenance*.

**M9. RUP-SE: rational unified process for systems engineering [32]**

RUP-SE is the systems engineering extension and adaptation of the well-known RUP SLDC for software engineering. It relies on the key concept of system as "… *a set of resources that provide services that are used by an enterprise to carry out a business purpose or mission. System components typically consist of hardware, software, data, and workers. Systems are specified by the services they provide, along with other non-behavioral requirements such as reliability or cost of ownership. Designing a system consists of specifying components, their attributes, and their relationships.*" [32].

RUP-SE is focused on designing generic engineering systems (which can or not include software). In this study, RUP-SE is considered explicitly for SoSS development. RUP-SE considers (i) modeling of system architecture, (ii) details concerning with all system components (hardware, workers, information components), this

| **RUP-SE modeling level** | **The level expresses:** |
|---|---|
| Context | The system and its actors. |
| Analysis | Initial partition of the system over each view point to establish the conceptual approach. |
| Design | Transformation of the analysis-level model into specifications of hardware, software and people. |
| Implementation | Realization of design model into specification for a specific configuration. |

Table 6. RUP-SE modeling levels (abstraction levels of modeling).

is to say, not just software concerning (iii) multiple teams in concurrent way, (not just individual or a single team work), and (iv) redesign of IT infrastructure to support evolving business-processes. RUP-SE proposes a suite of abstraction levels for modeling the system (RUP-SE modeling levels) shown in Table 6.

The modeling levels can be particularized by specific viewpoints such as enterprise, computation, engineering, information, and process. Trough these levels of RUP-SE modeling, the design process comes from the abstract to the physic layers (similar to the model transformation proposed by MDA). In each modeling level (context, analysis, design and implementation), a model is achieved for each viewpoint (enterprise, computation, engineering, information, process). Each pair modeling level and viewpoint is called a "system view", e.g., for the context level the contextual view of the enterprise is created, the logical-contextual view, the information-contextual view, the physical-contextual view, the process-contextual view, and so for the next two modeling levels. For the implementation level just two views are created (implementation-enterprise view and a second view of implementation that embraces computational, information, engineering and process viewpoint). For each view, one or more specific artifacts are proposed. RUP-SE is an extent of RUP that embraces several characteristics of modern systems, such as the need for an architecture and business orientation, but it is not focused especially on a service-oriented view.

*3.2 Comparison of the nine relevant 2000-2007 SoSE SDLCs*

Table 7 shows the comparison between the nine relevant SDLCs identified in core SoSS literature during the 2000-2007 period. The first column reports the macro phases. The second column captures the different SDLC phases. The third column focuses on SDLC artifacts based on MDA. The fourth column shows the business, architectural and application domains. The fifth column denotes the MDA-based SDLC activity. The remainder columns are for describing, and comparing, each SoSE SDLC (indicated as M1 through M9 for ease of reference).
The main findings of such a conceptual comparison are the following:

(a) M7, M8 and M9 are the most complete SDLCs but even these have "methodological gaps" when compared with the generic MDA-based SDLC (reported in Figure 5).

(b) The activity of business process modeling can be identified as mainly addressed by almost all of the SoSE SDLCs.

(c) M4 is the least complete SDLC of the nine SDLCs used in the comparison. However, this contributes with a skeleton for proposal system constructions.

(d) M1, M6 and M7 include the development phase in which service definitions are transformed into executable services in an executable BPEL4WS format. This is considered of high practical relevance because these SDLCs take advantage of the available computing service technology which has emerged before service-oriented methodologies.

(e) M5 is strongly based on MDA but still lacks some core activities identified in the generic MDA SDLC (e.g., enterprise architecture design, planning, system modeling (composition and orchestration).

(f) Out of the most complete SDLCs (M7, M8, and M9), M7 is the most innovative SLDC through the inclusion of analysis tools and techniques as "green-field, "top-down", "bottom-up", "out-of-the-middle", XML, WSDL and BPEL. However, it still fails to consider important activities such as requirements, business modeling, architecture, service orchestration, among others.

| MacroPhase | Phase | Generic Expected Artifacts | Domain | MDA-based SDLC | M1 (Karastoyanova, 2003) | M2 (Kotonya, 2004) | M3 (Ivanyukovich, 2005) | M4 (Kühne, 2005) | M5 (Cox, 2005) | M6 (Karakostas, 2006) | M7 (Papazoglou, 2007) | M8 (Qing, 2007) | M9 (Rational, 2003) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Definition | Requirements | CIM | Business | Business Process Model (BPM) | X | System Requirements Definition:<br>1.Requirements elicitation.<br>2.Requirements scope.<br>Requirements modeling. | Business Modeling | Structred analysis of requirements.<br>Requirements validation. | Business Process Model (BPM) | X | Scenario analysis.<br>Analysis and design of services.<br>Process identification | Market scan<br>Requirements Engineering<br>Business Modeling | Business Process Model (BPM) |
| | | | Architecture | Enterprise Architecture Design | X | X | X | X | X | X | X | X | System base-architecture model |
| | | | Application | Planning | X | X | Requirements | X | X | Service definition | Gap analysis | X | Requirements |
| Development | Design | PIM, PSM | Business | None proposed activity | X | X | X | X | X | X | X | Service Design | X |
| | | | Architecture | Solution Architecture Design | X | Mapping requirements to architectural design:<br>1.Dividing service descriptions in logic subsystems.<br>Establish subsystems interfase. | X | X | Definición SOA. | X | X | X | X |
| | | | Application | System Modeling<br>(Composition and Orchestration) | Modeling and assembling process models. | X | Analysis and Design | Transformation (orchestration) | X | Service definition and orchestration | Service specifications.<br>Process specifications. | Application Design | Modeling and assembling of process model |
| | Construction | Platform Executable Model | Business | None proposed activity | X | X | Integration | X | X | X | X | X | X |
| | | | Architecture | None proposed activity | X | X | Integration | X | X | X | X | Service Orchestration /Composition<br>Registry selection | X |
| | | | Application | Transformation<br>Deployment | Generating process definition.<br>Pre-processing<br>Deployment | System composition:<br>1.Replacing subsystem definition with components and services.<br>Adapting components and services. | Implementation<br>Test<br>Deployment<br>Integration | X | Service implementation and work flows.<br>Interfaces implementation. | Executable transformation | Code Web services<br>Code business process<br>Provisioning<br>Deployment | Service Development<br>Service Testing<br>Service publishing<br>Application Implementation<br>Module Testing | Implementation<br>Test<br>Deployment<br>Integration |
| Evolution | Operation | Evaluation & Evolution Plan | Business | Operative Evaluation | X | X | X | X | Operation, monitoring and optimization. | X | Execution<br>Monitoring | Service Management | X |
| | | | Architecture | Deployment | X | X | X | X | | X | | Registry maintenance | X |
| | | | Application | Deployment<br>Postmortem Analysis | Run-time<br>Post-execution time | Verification | Configuration management, Project management and Environment | X | Maintenance and growing. | X | X | Service monitoring<br>Application testing and maintenance | Configuration management<br>Project management<br>Environment |

Table 7. Comparative view of SOSE SDLCs.

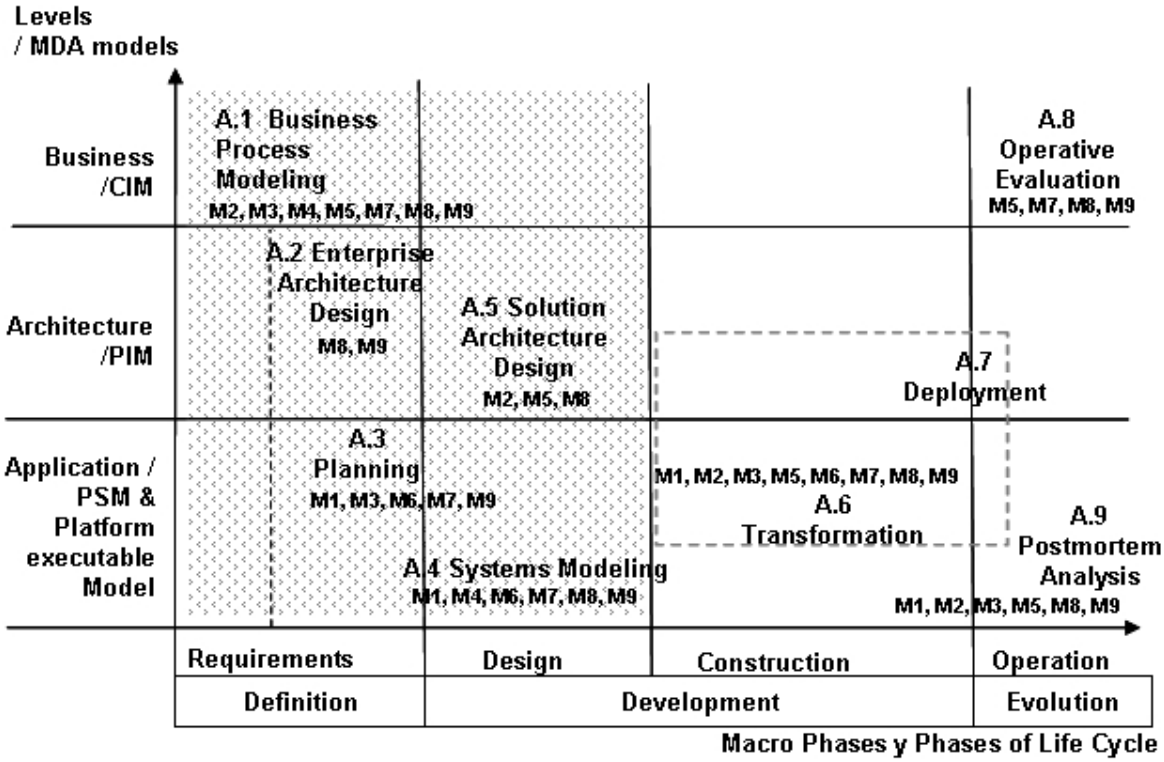


Figure 5. Generic MDA-based SDLC and the nine SOSE SDLCs.

Figure 5 complements this comparison showing the activities of the generic MDA-based SDLC included in each SoSE SDLC.

Explicit findings shown in Figure 5 are the following: (a) Some SDLCs (M1 and M6) are generally focused on the application domain, with little or null specific attention to the business and architectural domains. Similarly, their focus on the macro phases is primarily situated in the development macro-phase, and partially in the definition and evolution macro-phases; (b) M1 and M6 have a similar structure of activities, but in particular M1 considers a related post-mortem evaluation while M6 does not consider it; (c) M2 and M5 also have a similar structure of activities (and additionally these consider three levels in cascade) but none propose system planning activity, and (d) M7, M8 and M9 are the most complete SDLCs (their ID appears in almost the nine generic activities proposed in the generic MDA SoSE SDLC). Hence, this comparison has been based on the completion criterion compared with a generic MDA-based SDLC. Nevertheless, a rigor-agility balance analysis is still required.

*3.3 Comparison of the nine 2000-2007 SOSE SDLCs: agility – rigor view*

Table 8 shows the results of the rigor level assessment for each SOSE SDLC. The ordinal scale used is null, low, medium, and high, using scores from 0 to 3, where null is 0 and high is 3. Scores from 4 to 5 are reserved for traditional SDLCs which is congruent with Boehm and Turner [8] assessment. Thus, the maximum rigor score in this analysis for each SOSE SDLC is 45 points (15 items x 3 points).

To demonstrate the level of rigor and agility, M3 can be used as an example. Given that M3 is strongly based on RUP, its rigor can match that

| | Factor 1 | | | | | Factor 2 | | | | | Factor 3 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Level of Concerns** (+ rigorous = - assumptions) | | | | | **Life-Cycle Coverage** (+ rigorous = + coverages) | | | | | **Sources of Constraints** (+ rigorous = + restrictions) | | | | |
| | Agility + ASSUMPTIONS − Rigor | | | | | Agility − COVERAGE + Rigor | | | | | Agility − RESTRICTIONS + Rigor | | | | |
| **SDLC FOR SOSS** | **Business Enterprise** | **Business System** | **Multi-team project** | **Single-team project** | **Individual** | **Concept Development** | **Requirements** | **Design** | **Development** | **Maintenance** | **Management Processes** | **Technical Practices** | **Risk / Opportunity** | **Measurement Practices** | **Customer Interface** |
| **M1** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| **M2** | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 2 | 1 | 0 | 1 |
| **M3** | 0 | 2 | 2 | 2 | 0 | 3 | 2 | 3 | 2 | 2 | 2 | 1 | 2 | 1 | 2 |
| **M4** | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **M5** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **M6** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| **M7** | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 0 | 1 | 0 |
| **M8** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **M9** | 1 | 2 | 2 | 2 | 0 | 3 | 2 | 3 | 2 | 2 | 2 | 1 | 2 | 1 | 2 |

Table 8. Comparison of nine SOSE SDLCs: rigor-agility view.

proposed by Boehm and Turner [8] for RUP (i.e., medium to high agility level). From Table 8, the comparative level of rigor of each SoSE SDLC can be reported. Table 9 shows similar results (rigor level of each SoSE SDLC) but ordered and already classified. The rigor level is derived directly from the scores assigned in Table 8. Agility scores are calculated from the maximum rigor score (45 points) minus the rigor score reached by the SDLC under review. For instance, for SDLC M2, the rigor level score is 9, and the agility level score is 36 points (e.g. 45-9).

Finally, an assessment of the rigor and agility balance of each SoSE SDLC is reported in Table 9. Theirst column reports the SDLC. The second and third columns report the agility and rigor rates (total score divided by maximum score of 45 points) respectively. The fourth column reports the percentage range of balance (for example, for M9 whose values are 40 percent and 60 percent for agility and rigor levels, respectively.). Finally, in the fifth column, the rigor-agility balance Index is reported. Such index is calculated dividing the minimal percentage of rigor-agility balance by the maximal percentage of agility or rigor balance reached by each SDLC. For instance, for M9 the minimal and maximal percentages were 40 percent and 60 percent for agility and rigor balances, respectively.

| | Rigor – Agility | | | |
|---|---|---|---|---|
| SDLC for SoSS | Agility | Rigor | Rigor-Agility Balance (in %) | Rigor-Agility Balance Index min(R,A)/max(R,A) |
| M9 | 18/45 | 27/45 | 40%-60% | (40/60) = .67 |
| M3 | 18/45 | 27/45 | 40%-60% | (40/60) = .67 |
| M7 | 35/45 | 10/45 | 22%-77% | (22/77) = .29 |
| M2 | 36/45 | 9/45 | 20%-80% | (20/80) = .25 |
| M6 | 38/45 | 7/45 | 16%-84% | (16/84) = .19 |
| M1 | 39/45 | 6/45 | 13%-87% | (13/87) = .15 |
| M4 | 39/45 | 6/45 | 13%-87% | (13/87) = .15 |
| M8 | 39/45 | 6/45 | 13%-87% | (13/87) = .15 |
| M5 | 40/45 | 5/45 | 11%-88% | (11/88) = .13 |

Table 9. Rigor-agility balance assessment for the nine SoSE SDLCs.

Hence, the main findings from this rigor-agility balance comparison are as follows: (a) most of the SoSE SDLCs are focused on agility issues rather than on rigor; (b) there is a bias to unbalanced SDLC (e.g., more focus on agility or rigor) in most of the SoSE SDLCs; (c) the unbalanced bias is toward the agility issue; (d) no main traditional SDLC emerges as a common base (however, the most balanced SoSE SDLC (M3) is based on RUP); (e) main proposals are focused on the definition macro phase, the requirements phase, and the business domain; (f) while all of the SoSE SDLCs contribute to our knowledge of how to develop service-oriented software systems, all of them can be considered incomplete regarding the generic MDA-based SDLC; (g) new dimensions for comparing SoSE SDLCs have emerged to complement SDLC rigor, SDLC agility and SDLC rigor-agility balance levels (these new ones are SDLC completeness [19] and SDLC automation (CASE) support); and (h), M9 and M3 are the most balanced SLDCs.

## 4. Conclusions

In this study a conceptual description and comparison of nine relevant SoSE SDLCs identified in core SoSE literature during the 2000-2007 period has been presented. This comparison has been developed from (i) a MDA-based SDLC completeness level, and (ii) a Boehm-Turner's rigor-agility criterion. In the first case, three SDLCs (M7, M8 and M9) emerge as most complete. In the second case, only two emerge (M9 and M3). Combining both set of results, M9 emerges as the most recommended SoSE SDLC at the time of this study. From a practical viewpoint, these three criteria can be also considered for selecting a SDLC for developing SoSS: (i) the overall completeness and rigor-agility balance criteria (M9: a rational unified process for systems engineering (RUP-SE)) because it appears in both top lists), (ii) the agility preference criterion (M8: a stakeholder-driven SOA-LC model (SOA-LC – service oriented architecture life cycle model), or M7: BP-DLC (business process development life cycle), and (iii) the completeness criterion (M9 or M3: toward an SODM (*toward a service-oriented development methodology*).

We believe that this conceptual study contributes to SoSE discipline through the organization and methodological description and comparison of relevant contemporaneous (2000-2007 period) SoSE SDLCs reported in the literature. It also provides practitioners with an initial generic MDA-based generic SDLC and three criteria for selecting a SoSE SDLC. Before this study,

practitioners had to select a SoSE SDLC from a disperse variety of proposals or use a similar one to that used for non service-oriented software systems. Because SoSE transcends the previous OOSE paradigm, we hypothesize that OOSE SDLCs cannot be used directly in SOSE projects.

We consider also that a main limitation of this study is its high-level perspective given the vast quantity of conceptual information to be analyzed. However, the novelty of this scheme and the few overall comparisons reported suggest that this study serves as an initial comparison framework for identifying the current contributions from several proposed SDLCs as well as for detecting knowledge gaps. Finally, the main academic conclusion that emerges is that none of the nine SoSE SDLCs can be considered a standardized and totally accepted SoSE SDLC for academy and by extension for practitioners and thus, further conceptual and empirical research on SoSE SDLCs is recommended.

*Acknowledgements*

Authors thank Dr. Carol Pollard from Appalachian State University for his insightful recommendations for improving this paper.

## *References*

[1] Di Nitto E., Hall R.J., Han J., Han Y., Polini A., Sandkuhl K. & Zisman A., Report on the International Workshop on Service Oriented Software Engineering (IW-SOSE06), ACM SIGSOFT Software Engineering Notes, Vol. 31 No. 5, 2006, pp. 36

[2] Stiller E. & LeBlanc C., Project–based Software Engineering: an object-oriented approach, Addison-Wesley, 2001

[3] Kotonya G., Hutchinson J. & Bloin B., A Method for Formulating and Architecting Component and Service-oriented Systems, Computing Department, Lancaster University, LA1 4YR, 2004, pp. 1-23, UK.

[4] Rodríguez L.C., Mora M. & Alvarez F.J., A Descriptive/Comparative Study of the Evolution of Process Models of Software Development Life Cycles (PM-SDLCs), SIS 07: Simposio de Ingeniería de Software in ENC 2007: Encuentro Internacional de Computación, 2007, pp. 1-6, Morelia, México

[5] DeMarco T. & Boehm B., The Agile Methods Fray, IEEE Computer Society, Software Technologies, June, 2002, pp. 90-92

[6] Beck K. & Boehm B., Agility through Discipline: A Debate, IEEE Computer Society, June, 2003, pp. 44-46

[7] Boehm B., Get Ready for Agile Methods, with Care, IEEE Computer Society, Software Development, January, 2002, pp. 64-69

[8] Boehm B. & Turner R. Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven Methods, Proceedings of the 26th International Conference on Software Engineering (ICSE'04), 0270-5257/04, IEEE, 2004, pp. 718-719

[9] Avison D. & Fitzgerald G., Where now for development methodologies?, Communications of the ACM, 46(1), 2003, pp.78-82

[10] Williams L., A Survey of Agile Development Methodologies, North Carolina State University, Computer Science Department, 2004, pp. 209-227

[11] Amsden J., Modeling SOA IBM (PARTES 1,2,3,4,5), Level: Introductory, STSM, IBM, 2007

[12] Ruokolainen T., Service-Oriented Software Engineering: An Introductory Lecture, Toni.Ruokolainen@cs.Helsinki.FI, 2006

[13] Arsanjani J., Hailpern B., Martin J. & Tarr P.L., Web Services: Promises and Compromises, IBM Research Division, Thomas J. Watson Research Center, 2006, pp. 1-18

[14] Papazoglou M.P., Traverso P., Schahram D., Leymann F. & Bernd J., Service-Oriented Computing: Research Roadmap, Tilburg University, The Netherlands, e-mail: mikep@uvt.nl., 2006, pp. 1-29

[15] Zimmermann O., Krogdahl P. & Gee C., Elements of Service-Oriented Analysis and Design: An interdisciplinary modeling approach for SOA projects, IBM, 2004, pp. 1-17

[16] Mora, M., Gelman, O., Frank, M., Cervantes, F. & Forgionne, G., Toward an Interdisciplinary Engineering and Management of Complex IT-intensive Organizational Systems: a Systems View, International Journal of Information Technologies and the Systems Approach, 1(1), 2008, pp.1-24

[17] Dumas M., Towards a Semantic Framework for Service Description, Cooperative Information Systems Research Centre, 2000, pp. 1-20, Queensland University of Technology, Australia

[18] Rodríguez-Martínez L.C., Mora-Tavarez M. & Macías-Luévano J., La Integración de Servicios Informáticos: Un reto actual para el Cambio de las Organizaciones Modernas, XIV ACACIA, 2010, pp. 1-18, México

[19] Mora, M., Gelman, O., O'Connor, R., Alvarez, F. & Macias-Luevano, J., A Conceptual Descriptive-Comparative Study of Models and Standards of Processes in SE, F. Stowell and M. Mora Editors, SwE and IT disciplines using the Theory of Systems, International Journal of Information Technologies and the Systems Approach, 2008, pp. 156-184

[20] Korbyn C., Modeling Components and Frameworks with UML. ACM Communications Vol. 43, No. 10, 2000, pp. 31-38

[21] Hopkins J., Component Primer, ACM Communications Vol. 43, No. 10, 2000, pp. 27-30

[22] Rodríguez L.C., Mora M., O'Connor R., Garza L.A., Álvarez F.J., Durán H.A., Toward a Framework for Comparison System Development Life Cycles (SDLCs) in Service –Oriented Software Engineering (SOSE), GITMA 2009 Conference, June 14-16, 2009, pp.

[23] Miller J. & Mukerji J., MDA Guide Version 1.0.1, OMG,2003

[24] Rodríguez L.C., Mora M., Alvarez F.J. & Vargas Martín M., Process Models of SDLCs: Comparison and Evolution, M. Rahman Syed and S. Nessa Syed Editors, Handbook of Research con Modern Systems Analysis and Design Technologies and Applications, 2008, pp. 76-89

[25] Karastoyanova D., A Methodology for Development of Web Services-based Bussines Processes, Thechnische Universität Darmstadt, 2003, pp. 1-8

[26] Ivanyukovich A., Gangadharan G.R., D'Andrea V. & Marchese M., Towards a Service-Oriented Development Methodology. Department of Information and Communication Technology, University of Trento, 2005, pp. 1-10, Italy

[27] Kühne S., Thränert M. & Speck A., Towards a methodology for orchestration and validation of cooperative e-business components, Institute of Cybernetics at Tallinn Technical University, ISBN 9985-894-89-8, 2005, pp. 29-34

[28] Cox D. E. & Kreger H., Management of the service oriented-architecture life cycle, IBM SYSTEMS JOURNAL, VOL 44, NO 4, IBM Corporation, 2005, pp. 709-726

[29] Karakostas B., Zorgios Y. & Alevizos C.C., Automatic derivation of BPEL4WS from IDEF0 process models. Software & System Modeling 5(2), 2006, pp. 208-218

[30] Papazoglou M.P. & Van Den Heuvel W., Business Process Development Life Cycle Methodology, Communications of the ACM, Vol. 50, No. 10, 2007, pp. 79-85

[31] Qing Gu & Lago P., A stakeholder-driven Service Life Cycle Model for SOA, ACM IW-SOSWE'07, 2007, pp. 1-7, Dubrovnik, Croatia

[32] Cantor M., Rational Unified Process for Systems Engineering, Rational Brand Services IBM Software Group, 2003