Computer Science and Information Systems 16(1):19–44 https://doi.org/10.2298/CSIS170703035R

SOCA-DSEM: a Well-Structured SOCA Development Systems Engineering Methodology

Laura C. Rodriguez-Martinez[1], Hector A. Duran-Limon[2], Manuel Mora[3], and Francisco Alvarez Rodriguez[3]

[1] Tecnológico Nacional de México/I.T. Aguascalientes, Av. Adolfo López Mateos 1801 Ote., Bona Gens, CP 20256, Aguascalientes, Ags., Mexico
lrodriguez@mail.ita.mx
[2]CUCEA, University of Guadalajara, Periférico Norte 799, Zapopan, Jalisco, Mexico
hduran@cucea.udg.mx
[3] Autonomous University of Aguascalientes, Av. Universidad 940, Ciudad Universitaria, CP 20131, Aguascalientes, Ags., Mexico
mmora@correo.uaa.mx, fjalvarez@correo.uaa.mx

Abstract. Service-oriented Software Engineering (SOSE) is a software engineering paradigm focused on Service-oriented Computing Applications (SOCAs), for what SOCA development methodologies are required. Recent studies on SOCA development methodologies revealed theoretical and practical deficiencies. Thus, academicians and practitioners must adapt development methodologies from other paradigms or use the available partial SOCA development methodologies. Also, since the high acceptance of agile approaches, we claim new well-structured and balanced agility-rigor methodologies are required. Then, this paper proposes a new ***SOCA Development Systems Engineering Methodology***, including its description, the explanation of its theoretical foundations and the illustration of its use with a prototype of a running example. Two pilot empirical evaluations on usability metrics are also reported. Findings support both theoretical adequacy and positive perceptions from the evaluators. While further empirical tests are required for gaining more conclusive evidences our preliminary results are encouraging.

Keywords: Service-oriented Software Engineering (SOSE), Service-oriented Computing Application (SOCA), Model-Driven Architecture (MDA), RUP-SE, MBASE, agility-rigor balance.

1. Introduction

This study is located in the research area of Software Engineering (SE), in the part referred to as the Service-oriented Software Engineering (SOSE) paradigm, which can be seen as an evolution of the previous Object-oriented (OOSE) and Component-based Software Engineering (CBSE) paradigms [25], but that addresses loosely coupled distributed systems. The SOSE paradigm aims at producing high-quality Service-oriented Computing Applications (SOCAs). These SOCAs can be defined as software systems built through a suite of software services. Software services are auto-contained

computational work units provided by internal or external parties (service providers) to other parties (service customers) [37]. Software services have been also defined as "*autonomous, platform-independent entities that can be described, published, discovered, and loosely coupled in novel ways*" [50]. Different from the object- and component-oriented paradigm, SOSE focuses on coarse-grained distributed services that are loosely coupled. Nevertheless, both the object- and component-oriented paradigms are still helpful to construct independent services, which are the building blocks of a SOCA.

The SOSE research stream investigates "*systematic, disciplined and quantifiable approaches to develop service-oriented systems*" [25]. For this aim, SOCA engineering methodologies are required [37], [25]. In particular, [37] indicates that one of the core research challenges in the SOSE paradigm is defining "*development processes and methodologies for service-oriented systems".* Some SOCA development methodologies have been proposed such as [39][40][51]. However, these methodologies have the following limitations. Current methodologies emphasize elaborating on the system architecture, the use of specific technology for constructing services -like BPEL (Business Process Executable Languages), WSDL (Web Service Definition Language), Web Services-, as well as aligning on business only in the requirements Phase (some of them include a market study and requirements engineering). However, such methodologies do not consider the analysis and design of services, composition, orchestration, and choreography –i.e. regarding the latter, they do not facilitate the transformation of the representation of the system in the business domain to the final implemented application-. These limitations make it difficult to narrow the conceptual gap between the problem (or business) domain and the system implementation (or application) domain [18][61].

We believe the reason there is a lack of more recent methodologies for SOCA development is that only until recently it has been recognized by academicians and Industry that Service-oriented Computing (SOC) [50][69] is not only a fashionable technology but a real need. For instance, only in recent years there have been some reviews of SOCA methodologies [25][5][34]. Also, the increasing need for business-IT alignment [24] reflects the fact that SOCA development is only recently gaining more attention.

In this paper, we propose a **SOCA D**evelopment **S**ystem **E**ngineering **M**ethodology, called ***SOCA- DSEM***, that considers: (i) concerns of business, architecture and system (application) along the whole development process; and (ii) the analysis and design of services, composition, orchestration, and choreography. In addition, our proposal is a well-structured methodology meaning that it covers all the software system development life-cycle in a well-structured manner i.e. our methodology includes phases, activities, models, and products as well as roles and an iterative development process. Our methodology focuses on defining the structural activities, the execution order of these activities and their associated products. In the case of complex systems, our methodology can employ specific methods. This is similar to other methodologies or models such as RUP and Spiral that enable the use of specific methods. For example, a specific method regarding graphical user interface design can be applied as well as others more specialized methods like [7] that considers the architectural principles to develop SOCAs.

The design of our development methodology is theoretically underpinned by four core design building blocks: 1) Service-oriented Computing (SOC) [50][69], 2) Model-

Driven Architecture (MDA) [42], 3) two popular development systems engineering methodologies (RUP-SE [10] and MBASE [12]), and 4) recommendations for agility-rigor balance [9]. Currently, our approach does not have tool support to automate model-to-model transformations, rather, these transformations need to be performed manually.

We illustrate the use of our methodology with a running example consisting of a service-oriented system that performs a quality-assessment of academic-courses. We evaluated our proposed methodology by employing two empirical evaluations with a group of 15 international academicians on Software Engineering, and a group of 32 MSc partial-time students from an MSc course on an IT program in Mexico. In the first evaluation, the subjects reviewed the methodology for rating the level of theoretical validity. In the second evaluation, the subjects rated the levels of usefulness, ease of use, compatibility, result demonstrability, and behavioral intention of use. These results show both theoretical adequacy of our methodology and adequate scores on the perceived metrics collected from the MSc course on IT.

The remainder of this paper is structured as follows. In Section 2, the theoretical basis for the design of our methodology is reported. In Section 3, the methodology is described and its utilization is illustrated with a running example in Section 4. In Section 5, the results of the two empirical evaluations are reported. Finally, in Section 6, both the conclusions and recommendations for further research are presented.

2. Review of Theoretical Basis

This section describes the aforementioned theoretical basis-core design building blocks- of our proposed methodology namely, Service-Oriented Computing (SOC), Model-Driven Architecture (MDA), two popular development methodologies -RUP-SE and MBASE-, and recommendations for achieving agility-rigor balance.

2.1 Service-Oriented Computing (SOC). The implementation of the Service-Oriented paradigm is addressed by Service-oriented Computing (SOC), which can employ the Object-oriented and Component-based paradigms [69], [50] to build individual coarse-grained services. SOC approaches target loosely distributed systems and aims at building applications that integrate the interoperability of such coarse-grained services. Thus, SOC approaches involve the study and advances referring to building services in a way that multiple SOCAs can simultaneously use the same set of services [69]. The set of services are platform-independent for increasing the SOCA inter-operability [69]. The SOC paradigm also advocates providing services in different heterogeneous and external computational platforms [25]. Moreover, some SOCAs can dynamically re-configure their contracted services (e.g. a service provider can be replaced by an alternative service provider at runtime) [25]. Furthermore, the utilization of newer SOC programming technologies can be used for constructing SOCAs -e.g. we use the Service Component Architecture (SCA) hosted in a Java interface development environment for the running example presented in this paper [6].

A SOCA can be defined as a distributed and loosely coupled software system represented and constructed as a main customer control code calling a suite of computational services [3]. According to [16], services -as a broad concept- can be classified in *Business Services* and *Computing Services*. In turn, *Computing Services*

can be classified as a *Business Computing Service* when it implements a Business Service or as an *Information and Communication Technology Computing Service* when it provides a service to another software system. Based on these concepts, a SOCA can be also conceptualized as a composition of *Business Computing Services* [56].

2.2 Model-Driven Architecture (MDA). MDA [42] and Model-Driven Development (MDD) [18] are part of Model-Driven Engineering (MDE) [18]. They address the conceptual gap between the problem domain and the implementation domain [18] in the context of the transformation of several models that are constructed during the process of software development [61]. One of the main differences among MDA and MDD is that the former advocates the use of a general-purpose modeling language like UML and the Meta Object Facility (MOF) [49] for meta-modeling whereas MDD advocates employing domain specific languages instead [54]. In addition, both MDD and MDA study models at different abstraction-levels, but the MDA approach defines four core types of models based on four similar related views: 1) Computing Independent Model (CIM); 2) Platform Independent Model (PIM); 3) Platform Specific Model (PSM); and 4) Executable Platform Model (executable PM). The CIM focuses on the system requirements and its environment without any technical consideration of the target computing platform. The PIM focuses on detailed system operational specifications without involving a specific platform. The PSM focuses on the specific computational implementation and how the system uses it. The executable PM focuses on both the final runtime model and the specific parts and provided capabilities of the computational platform. Based on an analysis of the main MDA literature [42] - regarding service-orientation- the following design recommendations can be derived. A software product/service can be developed by employing several transformations and refinements from one model to other models. The models must foster the portability, interoperability, and reusability of the final software product/service. Moreover, a software product/service, should be developed by phased and iterative development engineering methodologies (by applying successive model transformations and refinements). Finally, a software product/service must be designed by using several types of models (i.e. it is unlikely we are able to design a software product/service by using only a single or few models).

This paper is based on our previous work where we defined a generic MDA-based development engineering methodology [56] that is derived from the main core SOCA literature. Such a core SOCA literature involves MDA principles [42], a Service-oriented Analysis and a Design approach [69], a set of software service conceptualizations [2], and a three-phased software development engineering macro model [55]. This paper extends and adapts such a generic MDA-based development engineering approach [56] in two dimensions. The first dimension regards the levels of abstraction of *Business, Architecture* and *Application*. The second dimension involves the phases *Requirements, Design, Construction* and *Operation* of a generic SOCA development engineering methodology [55]. Where, the Business level matches the CIM, the Architecture level matches the PIM whereas the Application level matches both the PSM and the executable PM.

Hence, our previous work provides theoretical insights for helping on the design of specific SOCA development engineering methodologies. For the practical purposes of this paper, we found that SOCA development activities are implicitly related one-by-one to a product that we can name with the same name of the activity, e.g. the activity A1 “Business Process Modeling” produces a “Business Process Model”.

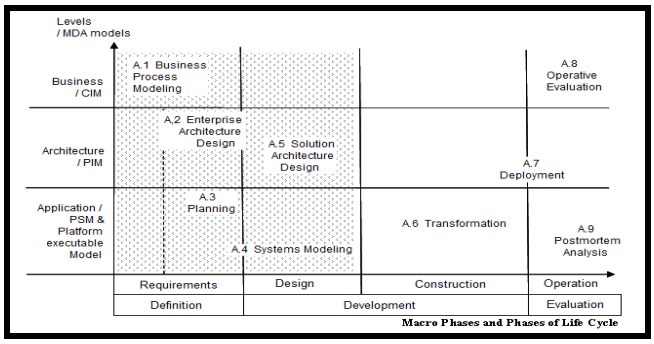


Fig. 1. Generic MDA-based Software Development Engineering Methodology for SOCAs (Quoted by [56])

Then, in order to define coarse-grained blocks of the structure of our SOCA methodology, from the MDA approach [56], we took the activities labeled as *A1..A9* (see Fig. 1) each one associated with a specific product. We also took from [56] the macro-phases *Definition*, *Development and Evaluation*, and the phases *Requirements*, *Design*, *Construction*, and *Operation*. At the same time, such an MDA approach [56] integrates some knowledge about software system development processes for SOCA, including RUP processes and promoting MBASE well-structured processes.

2.3 Software development engineering methodologies. Development methodologies combine the best practices from their predecessors [4], [55]. Two exemplary cases of these methodologies are the RUP for Systems Engineering (RUP SE) [10] and the Model-based System Architecting and Software Engineering (MBASE) [12]. They both include the flexibility of the iterative and incremental process and relevant activities for software projects like Risk Management and Business Modeling activities. RUP SE [10] is a methodology that is an enhancement of the well-known object-oriented or component-based RUP methods, and incorporates additional diagrams and techniques that are not reported in the standard RUP such as: context diagrams, system use cases with services, business process, process interaction diagrams, and system and sub-system architectures. RUP SE combines activities (that can be repeated in an iterative way), within phases that end with specific control milestones. The phases (Inception, Elaboration, Construction, and Transition) and the activities (Business Modeling, Requirements, Analysis and Design, Implementation, Test, Deployment, Configuration and Change Management, Project Management, and Environment Management) in the RUP SE are the same ones reported in standard RUP. In particular, the Business Modeling activity is mandatory and not optional as in normal RUP, and it is augmented with additional analysis diagrams [10]. Regarding MBASE, this methodology provides a system-based approach for developing integrated software systems [12]. MBASE is based on the Boehm's Win-Win Spiral Model [8] which proposes a system's specification with six elements: Operational Concept Description, System and Software Requirements Definition, System and Software Architecture Description, Life Cycle Plan, Feasibility Rationale Description, and Risk-driven prototypes. MBASE combines an incremental approach (with Inception, Elaboration, Construction, and Transition phases) with an iterative strategy (for developing the six core elements previously reported) for developing software systems. Similar to RUP SE, each phase ends with specific control milestones. In MBASE there are three core control milestones: Life-

Cycle Objectives, Life-Cycle Architecture and Plan, and an Initial Product ready for Operational use. In MBASE, the Business Modeling and Risk Analysis activities are also considered core activities and not optional ones as reported in other methodologies.

Hence, these two methodologies can be considered well-defined -according to the expected structure of phases, activities, products, roles and tools- [48], and well-balanced on agility-rigor issues [56]. Both methodologies are considered suitable for general purpose development. Therefore, for their use in SOCA development, they must be adapted. Nevertheless, we consider these two methodologies useful as a basis for generating new methodologies for SOCAs. Finally, we adopted the incremental and iterative philosophies of RUP and MBASE for favoring agility in the process. Specifically, from RUP we took the loops to control the development project, in particular inception, elaboration, transition, and deployment. Moreover, from RUP and MBASE we took a similar idea to milestones to implicitly control the project progress where the documents give evidence of the end of a stage or loop.

2.4 Agility-Rigor balance recommendations. Boehm and Turner [9] recommend employing an Agility-Rigor balance to the software development processes. These authors define a comparison framework that enables or facilitates the evaluation of software development processes. This framework defines three factors and divides each one in some characteristics, which can be measured for a software-system development life-cycle. Each characteristic can be assessed with the following values: non-existent, medium, and high. This indicating that a characteristic is not covered, partially covered, and fully covered, respectively. For an agility-rigor balance it is expected the assessed values to be in average medium (i.e. partially covered). Such factors and characteristics are the following:

*Factor 1. Level of Concerns***.** This factor identifies the organizational scope of use for which a development life-cycle proposes specific guides for its use (not assumptions). Five possible concerns are the following. The *Business Enterprise concern* indicates us whether the methodology can be used within the organizational boundaries (not necessarily with co-located teams) and whether the methodology provides guidance to potentiate the inter-team communication by suggesting information sharing strategies, use of communication tools, and ways to gradually introducing rigor in project work in order to support distributed development. The *Business System concern* indicates if the methodology is easier to be applied to business systems than to engineering or scientific applications. The *Multiteam Project concern* indicates if the methodology can be used by co-located teams. The *Single-team* concern indicates if the methodology can be used by a single team located in a single office building. Lastly, the *Individual Project concern* indicates whether the methodology can be used by a single individual. The less restrictions we have the more agility we obtain.

Factor 2. Life-Cycle Coverture. This factor identifies the life-cycle (structural) activities for which the methodology proposes specific guidance. This factor evaluates the life-cycle coverture. The activities to evaluate it are: Concept Development, Requirements, Design, Development, and Maintenance. For each one it is verified if the life-cycle activities are covered by the development process of the methodology under evaluation. Here a more complete coverture is considered to have more rigor, i.e. less agility.

Factor 3. Sources of Constraints. Each development life-cycle proposes constraints to the implementer. When such a life-cycle is less constrained, it is considered more agile. Five possible sources of project-areas constraints are considered by this factor:

management processes, technical practices, risk/opportunity considerations, measurement practices, and the customer interface.

Boehm and Turner [9] employed such a framework and evaluated RUP and other methodologies (that are either considered agile or disciplined), finding that RUP is a good example of agility-rigor balance, i.e. RUP is balanced on agility-rigor [9]. Because of this, we took RUP as a reference. Then, we included an agility-rigor balance in our methodology by considering the three factors - *level of concern, life cycle coverture*, and *source of constraints*- defined by [9]. Regarding the level of concern and life cycle coverture factors, our methodology, similar to RUP, partially covers all stages of the development process and proposes local work for both single and multiple teams. With respect to the source of constraints factor, RUP partially covers enterprise restrictions, technical-practices restrictions, and measurement-practices restrictions. Moreover, customer restrictions are not covered whereas risk/opportunity restrictions are fully covered. On the other side, since our methodology focuses on the development process, our methodology fully covers technical-practices and customer-interfaces restrictions. Nevertheless, in order to promote agility, our methodology does not cover risk/opportunities or measurement-practices restrictions (which mainly focus on the management process).

3. Structure of the proposed SOCA Methodology

Our methodology defines the structure of the development-process [62] i.e. such a methodology defines the components (phases, activities, artifacts, and roles) [48], and the way to iterate. The roles are responsibilities, assigned to agents (a person or teams), involving the activities of the processes. As said earlier, our methodology is based on the MDA approach, thus it is required a set of "models" as part of the structure of the methodology. The models in the methodology involve a set of products [10]. We use some specific notations to construct the artifacts (i.e. products) in our running example, however, the methodology enables the use of any other notations. In general, our methodology includes structural activities, but also, some umbrella activities [62] like planning and evaluation are considered.

The steps of our methodology are divided in four phases:

1. Requirements. This phase includes activities related to CIM modeling, CIM evaluation, and life cycle planning.

2. Design. This phase concerns PIM modeling, PIM evaluation, PSM modeling, and PSM-evaluation.

3. Construction. This phase includes the activities of construction planning, executable PM modeling, evaluation, and deployment.

4. Operation. The activities included in this phase are evaluation planning, evaluation of the executable PM operation, results specification, and analysis for evolving the executable PM.

Our methodology involves a number of model-to-model transformations, which currently have to be carried out manually. Importantly, our methodology is not constrained to specific tools to build the artifacts. Each one of the phases is described in turn below.

As said earlier, other methods can be employed within our methodology. For example, for developing the system architectural design we can apply the method in [7]. Other examples involve methods that target specific software engineering areas such as requirements engineering and human computer interactions. In any case, our methodology gives explicit guidance on the activities that need to be carried out as well as their associated artifacts, this independently of the particular methods being used.

3.1 Requirements. In the Requirements phase, the entire development process is planned, the general requirements and the scope of the system are defined, and the expected system delivery is also planned. In the Requirements phase, a requirement elicitation is carried out, the objectives of the software system are specified based on requirement agreements, and finally, both the software systems scope and the expected software systems delivery are planned. The activities involved in this phase are the following:

1. CIM Modeling. The CIM (according to our SOCA Development Systems Engineering Methodology) corresponds to the Business Process Model. In this activity, the elicitation and specification of general requirements are elaborated for creating the Computation Independent Model (CIM). This activity generates the following products: (i) Service Specification that includes the Process and work flows specification and the Specification of the Services and their conceptual definitions, and (ii) Data Semantic Model of the Enterprise architecture that aims at defining data to narrow the gap between the business and technical domain. The suggested artifacts for representing the CIM are: System Context Diagram, Work System Snapshot, System Responsibility Table, Business Process Diagram, and Enterprise data view.

2. CIM Evaluation. In this activity, several stakeholders participate for detecting and correcting errors in the previous activity.

3. Life Cycle Planning. In this activity, a Life Cycle Plan is elaborated. Such a plan must contain: (i) a system scope and delivery plan, and (ii) a development plan.

At the beginning of the execution of the CIM Modeling activity of our methodology, we can apply any specific method for requirements elicitation that we can take from the requirements engineering research area such as [62][64]. Next, we can construct the specific products of the CIM.

3.2 Design. In the Design phase, the software system is designed, which implies the transformation of the CIM into the PIM takes place, where the PIM realizes the Solution Architecture of the System. The transformation of the PIM into the PSM is also carried out in this phase. The activities of this phase are the following:

1. PIM Modeling. In this activity, the CIM into PIM transformation is executed as the design of the Solution System Architecture. This activity must generate the following products: (i) Service Provider Architecture Design that also contains the definition of services operations; and the specification of the communication among services (service orchestration); (ii) Service Consumer Architecture Design that also contains processes with a business semantic specification and the workflows specification (service choreography); and (iii) the Database Design. The suggested artifacts for representing the PIM are: Service model diagram, Join Realization Table that represents the Service-Choreography, Service-Orchestration Definition, and the E-R System data model.

2. PIM Evaluation. In this activity, the PIM is reviewed to identify and correct critical errors.

3. PSM Modeling. In this activity, the PIM into PSM transformation, for defining the System Model, is realized. The PSM is composed of the products: (i) Service Provider Design that contains the service specifications of implementations -definition of service components, service design and service interfaces-, (ii) Service Consumer Design that contains the definition of the technology (languages) to be used for the implementation of the executable PM, and their collaboration, (iii) design of the database scheme, and (iv) User interface design. The suggested artifacts for representing the PSM are: Creation script of the database scheme in a Data Definition Language (DDL), Data Access Definition (e.g. XML-based, JDBC), Service realization diagrams (interfaces, operations = UML interface stereotypes + component-based specification of realization), and User interface design.

4. PSM Evaluation. In this activity, the PSM is reviewed to identify and correct critical errors.

At the end of the PIM Modeling activity we can employ a specific architectural design method such as [7][13]. Regarding user interface design, which is part of the PSM Modeling activity, we can use guidelines or methods such as [15][22] proposed in the human computer interface knowledge area.

3.3 Construction. In the Construction phase, the software system is constructed, and released, this involves the transformation of the PSM into the Executable Platform Model (Executable PM). The activities of this phase are the following:

1. Construction Planning. In this activity, a plan for transforming the PSM to the executable PM is elaborated.

2. Executable PM Modeling. In this activity, the PSM is transformed into the executable PM, and each service along with its orchestration is implemented and tested. The artifacts suggested to use in this phase are: Services implemented and orchestrated, Services composed and published with a Service Component Architecture, and Choreography of services in work flows with HTML.

3. Executable PM Evaluation. In this activity, the executable PM is evaluated to detect and fix errors.

4. Deployment. In this activity, the executable PM is released.

3.4 Operation. In the Operation phase, the software system (the executable PM) is monitored and evaluated. Hence, monitoring and evaluation logs are analyzed for correcting unexpected problems and for functionality improvement or functionality increment. The activities of this phase are the following:

1. Evaluation Plan. In this activity, a plan is elaborated for evaluating the software system operation.

2. Evaluation of the executable PM operation. In this activity, the software system operation is monitored and evaluated.

3. Results Specification. In this activity, the evaluation of the software system is registered.

4. Analysis for evolving the executable PM. In this activity, the results are analyzed to make decisions for the purpose of system evolution.

3.5 Description of Roles. In our methodology, the roles are defined as development teams. Similar to RUP-SE's role teams, each one of them has a project administrator and a technical leader. Table 1 shows the roles of our methodology.

Table 1. Description of Roles of the SOCA Development Systems Engineering Methodology

Team Roles	Main Responsibility
Enterprise modeling	To conduct the Requirements phase activities. To elaborate the CIM MDA model.
System architecture	To conduct the high-level Design phase activities. To elaborate the PIM MDA model.
System and Services modeling	To conduct the low-level Design phase activities. To design the composition of services. To elaborate the PSM MDA model.
System and Services development	To conduct the Construction phase activities. To elaborate the composition of services. To elaborate the executable PM MDA model.
Deployment and Operation	To conduct the Operation phase activities. These activities must set the system in operational mode, perform required user training, and guarantee the expected functionality.
Evaluation and Evolution	To conduct the Support loop activities. These activities must guarantee the continuous operation of the system. Such activities are also responsible for monitoring, evaluating, and evolving the system.

Each role is also responsible of the administration of its own activities and phases. However, an administration team is posed for coordinating all the administration activities and ensuring their execution. The teams "System and Services development", "Deployment and Operation", and "Evaluation and Evolution" can constitute the development team. For large-scale systems it is necessary to have one team for each role. In the case of small/medium-scale systems, there can be a person per team, or even a few people performing multiple roles.

3.6 The incremental iterative approach. Our methodology performs iterations like in RUP, as shown in Fig. 2. This figure reports phases and loops and the stages -as in RUP- of Inception, Elaboration, Construction, Transition, and Support [62]. Also, similar to RUP, the colored shapes represent the possible amount of effort required for executing each phase of the process. Such a representation of the iterative way of our methodology implies that the construction of the models is built in an incremental way.

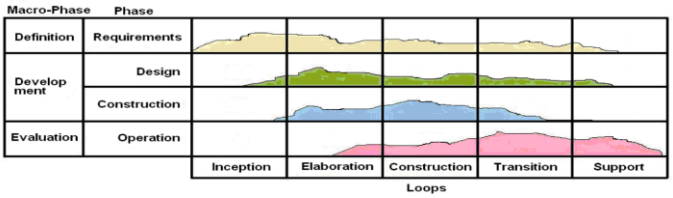


Fig. 2. The Iterative Model of the SOCA Development System Engineering Methodology

3.7 Model-to-model transformation. Our methodology defines three lines of model-to-model transformations. Such lines are represented in Fig. 3. The first artifacts constructed are System Context Diagram (D1) and the Work System Snapshot (D11) that are transformed into the artifacts System Responsibility Table (D12), Business Process Diagram (D7), and Enterprise data view (D13) each one in a line of transformation. Then the CIM is transformed into the PIM as follows. The System Responsibility Table (D12) is transformed into the artifacts Service model diagram (D2) and Service-Orchestration definition (D10). The artifact Business Process Diagram (D7) is transformed into the artifact Join Realization Table (D5), and the artifact Enterprise data view (D13) is transformed into the artifact ER System data model (D3). Next, PIM artifacts are transformed into PSM artifacts as follows. In the orchestration line, the artifacts Service model diagram (D2) and Service-Orchestration definition (D10) are

transformed into Service realization diagrams (D9). In the choreography line, the artifact Join Realization Table (D5) is transformed into the artifact User interface design (D8). In the data line, the artifact E-R System data model (D3) is transformed into the artifacts Creation script of the database scheme (D4) and Data access definition (D6). Finally, the PSM is transformed into the executable-PM as follows. In the orchestration line, the artifacts Service Realization Table (D9) and Service Orchestration Definition (D10) are transformed into the artifacts Service implemented and orchestrated in Java (D14) and Service composed and published with SCA (D15). In the choreography line, the artifact User interface design for HTML (D8) is transformed into Choreography of services in workflows with HTML (D16).

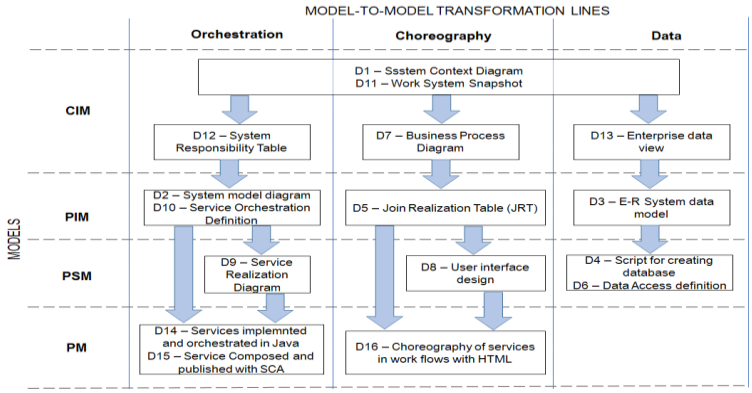


Fig. 3. Model-to-model transformation lines of the SOCA Development System Engineering Methodology

4. Motivating Example

We use a running example to illustrate the use of our methodology. Our running example involves a service–based system that supports the quality-assessment of biannual academic-courses. At the beginning, the system enables the teachers for elaborating two course-plans: "didactic instrumentation" that indicates the themes of the course and the didactic (teaching and learning) activities to be executed throughout the semester, and the "scheduled advance" that contains the planned dates to cover and to evaluate the course themes -or learning units of the course-. At three different audit trail moments, the teacher captures tracing data as audit trail of his/her work i.e. student-grades and real dates vs. planned dates. For each audit trail moment, the plans and tracing data are audited through the system by the department chair (chief) and the chief assistant. At the end of the semester, the system generates a final statistical report about the tracing data. In the running example, we illustrate the elaboration of the MDA-based models that conform to the artifacts of our methodology. More specifically, we report some of the artifacts that result from modeling the CIM, PIM, PSM and Executable PM.

4.1 Computation Independent Model (CIM). We present the artifacts System Context Diagram (see Fig. 4) -that represents the total context of the projected system-, and the

Business Process Diagram (see Fig. 5). The left part of the System Context Diagram artifact represents an overview of the business process, represented as a list of core elements: management forms, processes, and roles.

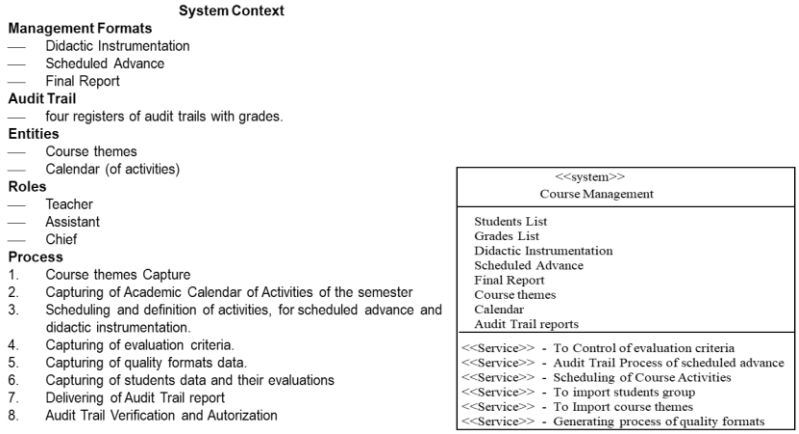


Fig. 4. CIM-D1 System Context diagram

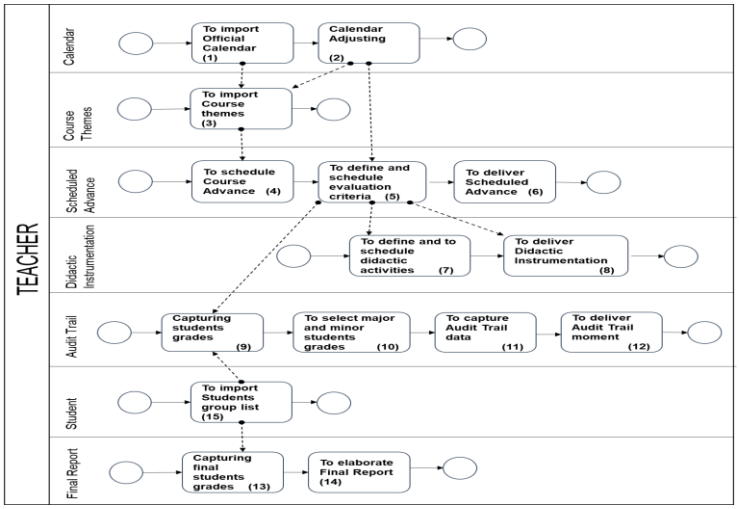


Fig. 5. CIM-D7 part c – A detailed Business Process Diagram

A Business-Process Diagram represents: (i) the activities of the private processes for each role -“CHIEF”, “TEACHER” and “ASSISTANT”-, where each private process is presented in a line; (ii) the interaction of public processes for the roles, that are presented at a high level of abstraction in a line per role; and (iii) diagrams of lower level of detail are constructed to show the specific workflows for each role for the specific cases where a more explanatory description is required. For the first increment of the development process (available in [57]), we only elaborated private processes of the role “Teacher”, the public processes collaborations of “Teacher” with the other roles, and the detailed workflow of the role “Teacher” (Fig. 5). Such diagrams present

the Business Processes from a low to a high level of detail based on the specific needs of the business requirements for any projected system. For example, the service "Calendar" in Fig. 6 has three operations: 2.1, 2.2, and 2.3 which are used for executing the sub-processes or activities 1, 2, 4, 5, and 7 of the detailed business process diagram (Fig. 5). The service "Audit Trail" has six operations labeled from 5.1 to 5.6.

4.2 Platform Independent Model (PIM). The PIM is the first model that is built during the first part of Design phase. The detailed Business Process Diagram presented in Fig. 5 is transformed into a Service model diagram (see Fig. 6), the service choreography -this one represented with Join Realization Tables, which are similar to use cases (available in [57])-, and the Service Orchestration Definition (see Fig. 7 a). The sub-processes or activities in the detailed Business Process Diagram are transformed into operations of the services in the Service model diagram. Also, these operations are grouped into components. Then, a Service orchestration definition (see Fig. 7 a) is elaborated with their implementations and interfaces. Interfaces (i.e. the operations the service component provides to other components) are represented on left side. Similarly, the component references (i.e. the services a component needs to call) are defined on the right side (see Fig. 7 b).

One component corresponds to one service (business service), and each service can have one or more interfaces. For each interface there is one implementation, and each implementation includes one or more operations of the corresponding service.

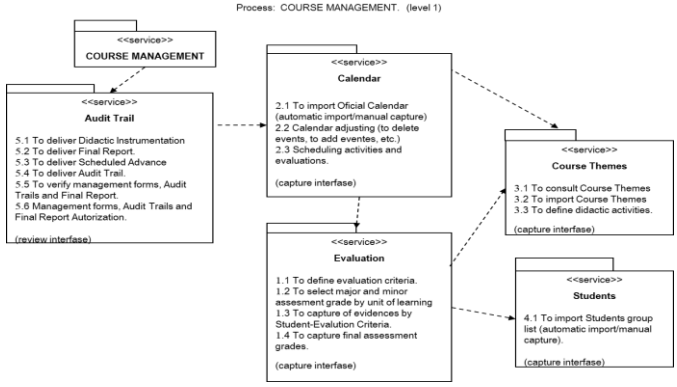


Fig. 6. PIM-D2 Service model diagram

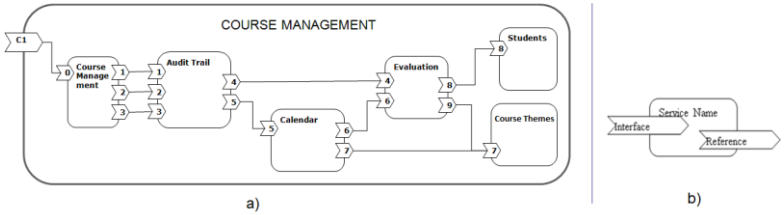
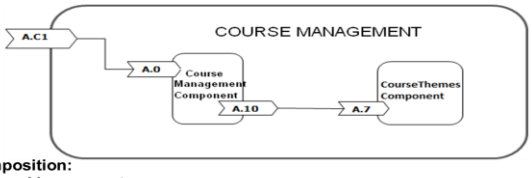


Fig. 7. a) PIM-D10 – Service Orchestration Definition-, **b)** Service Component with one Interface and one Reference

4.3 Platform Specific Model (PSM). The PSM is the second model constructed during the Design phase of our methodology. The Service orchestration definition and its

corresponding Service model diagram are transformed into the corresponding Service Realization diagrams (available in [57]), which describe the service and its realization with UML diagrams (i.e. one Service Realization Diagram is constructed for each service component in a Service Orchestration definition as shown in Fig. 7a). Also, JRTs are transformed into the user interface design (available in [57]).

4.4 Executable Platform Model (Executable PM). In our methodology, the Executable PM is built during the Construction phase. Here, Service orchestration definition and the Service orchestration diagrams were transformed into the artifacts of the Executable PM involving the programming code of the system prototype. Such a programming code is available in [57]. We implemented the main user interface of our running example in HTML. We also implemented the operation 3.1 "To consult Course Themes" (see the JRT in [57]) of the service "Course Themes" (see Fig. 6). Such an operation is used by the first step of the running example that regards the activity 3 "To import Course Themes". This activity is located in the private process (see Fig. 5). In the application domain, the prototype involves only a service composition of the subset of the Service Orchestration Definition presented in Fig. 7 a. Such a subset is shown in Fig. 8.



Service Composition:
(C1) Course Management

Service Components.
- CourseManagementComponet (just to publish the service composition, it does not have implementation)
- CourseThemesComponent

Service Interfaces = defines service implementation and interfaces.
(A.7) CourseThemesService + CourseThemesInterface .– Course Themes (this service interface is exactly how is presented in PIM-D10)

Service References
(A.10) CourseThemesReference .– Course Themes (this reference, in PIM-D10 is going to two service components. But, in this case it is going direct to CourseManagementComponent, because is the only operation to program in this first prototype).

Fig. 8. Fragment of Service-Orchestration definition of the implemented executable PM

We implemented the service "Course Themes" with a component (service). We defined code in SCA that orchestrates the service composition "Course Management", the service component "CourseThemes", and defined the reference "CourseThemeReference". The functionality of the services was implemented in Java. The service "CourseThemes" was implemented with both the class "CourseThemesService" and the interface "CourseThemesInterface", as shown in Fig. 8. The method "TemaItem[ ] get(String idMateria)" of the interface of the service "Tema" implements the operation 3.1 of the Service model diagram. The code of these programs can be obtained from [57]. Finally, some minor details regarding the implementation are the following. The service component "Course Management", the service component "CourseThemes", the reference "CourseThemesReference", the Java class "CourseThemesService", and the Java interface "CourseThemesInterface" correspond to "gestion", "Tema", "tema", "TemaImpl", and "Tema", respectively, in the implementation files.

5. Empirical Evaluation of the proposed SOCA Methodology from a Panel of SOCA Academicians and Practitioners

According to [44] a theoretical validation of a conceptual artifact can be also conducted with a similar *Face Validity* approach used in the simulation research stream [58]. Also, in other relevant disciplines a panel of experts is used for making decisions through a systematic process named Delphi method [47]. Furthermore, this evaluation method has started to be used for exploring a few relevant research problems in the discipline of software engineering [4], [26]. Thus, we applied this empirical approach as a preliminary evaluation of our work. The utilization of Likert 5- and 7-point scales have been reported as adequate regarding reliability and validity measures when the researcher wants to eliminate the neutral option [1]. In our research, we used for the first instrument a 5-point Likert scale because it is a new instrument and a moderate fine scale is recommended. In the second instrument we used a 7-point Likert scale because it has been extensively tested [31] and it has a finer scale.

5.1 Empirical Evaluation of proposed SOCA Development Systems Engineering Methodology from a Pilot Group of Software Engineering Academicians

A group of 15 software engineering academicians were contacted for evaluating the overall theoretical validity of the design of our methodology. Such evaluators were selected under the following criterion: to have a PhD level and at least 5 years of research-teaching experience on software engineering methodologies or software engineering professionals with at least a MSc student level and at least 5 years of professional expertise in software engineering methodologies. Most of them are from Latin America (LA) and two from Europe. This group was contacted through direct email contacts from the research team. Table 2 reports the specific demographic data of this panel of experts.

Table 2. Demographic Data of the Panel of Experts

Id. Eval	Academic Level of Evaluator	Expertise years in Software Engineering		Organization Type	Region
		Academic Settings	Industrial Settings		
1	PhD	5 – 10	< 5	Higher Education	LA
2	PhD	> 10	< 5	Higher Education	LA
3	PhD	> 10	------	Higher Education	LA
4	PhD	> 10	5 – 10	Government agency	LA
5	MSc Student	-----	5 – 10	Government agency	LA
6	PhD	> 10	< 5	Higher Education	LA
7	PhD	> 10	5 – 10	Higher Education	LA
8	PhD	5 – 10	< 5	Higher Education	LA
9	PhD Student	5 – 10	> 10	Government agency	LA
10	PhD	> 10	------	Higher Education	LA
11	PhD	> 10	< 5	Higher Education	LA
12	PhD	5 – 10	< 5	Higher Education	Europe
13	PhD	----	> 10	Higher Education	Europe
14	PhD	----	5 – 10	Higher Education	LA
15	MSc Student	5 – 10	> 10	Government agency	LA

The artifacts that were evaluated by the panel of experts were: 1) the user manual of our SOCA methodology, and 2) the documentation generated as a proof of concept for our running example. The 15 evaluators rated 6 items on theoretical validity (see Table

3). Each item was rated using a 5-point Likert Scale (from 1 for a total disagreement statement to 5 for a total agreement). The evaluation had a mean and standard deviation of **4.66** and **0.62,** respectively. Such results suggest that the panel of experts on software engineering methodologies considered our SOCA methodology as a theoretically supported development methodology.

Table 3. Evaluation of SOCA Development Systems Engineering Methodology from the Panel of Experts

Evaluated Item	Mean	Std. Dev.
P1. The conceptual model is supported by robust theoretical principles.	4.60	0.52
P2. The theoretical principles used to develop the conceptual model are relevant for the topic under study.	4.90	0.32
P3. The revised literature to develop the conceptual model does not present important omissions.	4.60	0.84
P4. The conceptual model is logically coherent.	4.70	0.67
P5. The conceptual model is adequate for the pursued purpose.	4.50	0.71
P6. The conceptual model provides a real contribution and it is not a duplicate of an existent model.	4.60	0.70
TOTAL	4.66	0.62

5.2 Empirical Evaluation of the SOCA Development Systems Engineering Methodology from Pilot Group of Software Engineering Practitioners

According to [27][35] a software development methodology can be evaluated through a *survey method* (also named field study). In particular in [35] it is reported a generic evaluation method named DESMET, where the survey method is one of the three ones suggested to be used. A survey method [35] "*is the collection and analysis of data from a wide variety of projects. The data collected in a survey are not as controlled as those obtained from a formal experiment but they can be analyzed statistically to identify important trends*".

Our methodology was evaluated through this *survey method*. In this survey took place a pilot group of 32 partial-time MSc students enrolled in a 2-year graduate program (23 enrolled in 2008-2009 period and 9 in the 2010-2011 period) in a Mexican public university located in the central region of Mexico. The first author taught a 4-weekeend graduate course on Software Oriented Architecture (SOA)/SOSE Development to this pilot group. These 32 partial-time MSc students (with a total of 64 class-hours per course) learned four main thematic units: 1) SOSE foundations, and its relationship with SOC and SOA; 2) SOC and SOA foundations, available technologies and the concept of SOCA or service oriented systems; 3) SOA/SOSE methodologies including a review of how service oriented systems can be implemented, and the methodology SOCA-DSEM; and 4) using SOCA-DSEM and SCA for constructing a SOCA example. For the End Term Project of this graduate course the students were asked to apply our SOCA methodology for developing a SOCA prototype of a real problem selected by themselves. The training to use the methodology included showing and explaining the complete running example that is illustrated in this paper.

The students were given a 2-week period for elaborating the project in teams of 4 people (to apply the methodology each team member took all team roles of the SOCA-DSEM). A total of eight teams were formed. Because this evaluation method does not qualify as experimental, the team formation was decided by the same MSc students. Each team selected the SOA technology to from the three technologies presented in the

second thematic unit of the course, namely Java Beans in J2EE and Servlets, C# in .NET using compositions in BizTalk Server with Business Activity Monitoring, and SCA for PHP platforms. The services were published as web services –i.e. remote component services-. Specifically, four of the teams used Java Beans with Servlets, two teams employed C# in .NET with BizTalk Server and Business Activity Monitoring, and two teams used SCA for PHP. The students only implemented Business Computing Services -also called "SOA Process Services"- as remote components, other kind of services such as Information and Communication Technology Computing Services -also called "SOA Infrastructure services"- were not implemented because SOCA -as we said earlier in subsection 2.1- is conceptualized as a composition of Business Computing Services [56].

The aforementioned technologies provide a way to publish the services in the web [59]. There are more sophisticated technologies and standards such as SOAP, WSDL, UDDI, RESTful Web Services among others that are useful to develop services [52][59][19]. However, these technologies were not used by the students as the technologies they employed are easier to learn and use in small projects. After this 4-weekend graduate course and the elaboration of the end term project, we measured the MSc perceptions on our SOCA development methodology by using the following constructs [31]: usefulness, ease of use, compatibility, result demonstrability, and intention of use. The questionnaires that were employed can be obtained from [57]. The demographic data of these 32 MSc part-time students is reported in Table 4. Finally, although the time of training involved only four weeks, the empirical evaluation was possible given that the students had already programming experience with the technologies they employed. On the other hand, our methodology is focused on the decomposition / composition of the services over distributed components, which can be implemented in three possible types of technologies, namely web services, Representational State Transfer (REST) approach, and messaging systems [7]. Despite the groups of students were small and there was limited time for developing the SOCA example, the students were able to build systems whose basic functionality worked correctly; this due to their previous programming experience with the employed technologies.

Table 4. Demographic Data of the Pilot Group of 32 MSc Part-Time Students

Demographic Variable	Highlights
IT background	All of them come from a BSc. in IT.
Age range	The sample contains similar groups (1/3) on the ranges of 26-30, 31-35, and 36-40 years.
Formal training in SwE	According to the MSc curricula, all of them (100%) were enrolled in at least 3 courses in SwE themes.
Main working role	Most of them (85%) are located in IT technical positions. Very few of them (15%) are located in managerial positions.
Scope of working organization	80% of them are working for organizations with a national scope and 10% are working for organizations with a worldwide scope.

According to [68], a construct is a theoretical concept, which cannot be measured directly, but through operational variables (named also items). We selected this set of constructs (*Usefulness, Ease of use, Compatibility, Result Demonstrability, and Behavioral Intention of Use*) because such constructs have been suggested elsewhere [53][46][65][45][36][66][30][41][11][21] as highly suitable for predicting the overall acceptance or rejection of a new designed artifact. Table 5 shows the constructs, their definitions, their operationalization, and their reported reliability measures from [31]

that reports the specific items for each construct C*i* where *i* goes from 1 to 5, thus, representing the 5 constructs we employed.

Table 5. Summary of Constructs used for Empirical Pilot Evaluation through a Survey Method

Construct	Definition	Operationalization	Reported Reliability
Usefulness	*the degree to which using IT innovation is perceived as being better than using the practice it supersedes.*	4-item 7-point Likert Scale	0.90
Ease of use	*the degree to which using a particular system is free of effort.*	3-item 7-point Likert Scale	0.90
Compatibility	*the degree to which adopting IT innovation is compatible with what people do*	3-item 7-point Likert Scale	0.88
Result Demonstrability	*the degree to which the results of adopting/using IT innovation are observable and communicable to others*	3-item 7-point Likert Scale	0.76
Behavioral Intention of Use	*the individual's intention to adopt or continue to use the IT innovation*	2-item 7-point Likert Scale	0.90

To assess the acceptance level from the pilot group of the 32 Part-Time MSc students, we planned initially to apply a one-tailed single-sample *t* Test [60] with an alpha value of 0.05 for each one of the five hypotheses (i.e. each one for each construct). First for supporting the normality assumption it was required to apply the single-sample t Test. Hence, we applied the Shapiro-Wilk test [43] to each 32 data sets for the 5 constructs. We found that the normality assumption cannot be supported, then we applied an alternative non-parametric test: the Wilcoxon signed-ranks test [60] for a single sample. There is another Wilcoxon test (Wilcoxon matched-pairs signed-ranks test) but it is only suitable for two dependent samples, and it was not our research case. Based on [60] a two-tailed bidirectional Wilcoxon signed-ranks test was used to evaluate the next null hypothesis "*does a sample of n subjects (or objects) come from a population in which the median value is equal to a specified value?*". In our research, we need to perform the test against a particular median value (fixed by the researchers) regarding whether it is less or equal to the fixed value. Thus, we use the one-tailed unidirectional test, and the null hypothesis was established as "*does a sample of n subjects (or objects) come from a population in which the median value is less or equal than a specified value?"*. This statistical non-parametric test has been used frequently in empirical software engineering research [38][17][29]. Thus, we used the one-tailed Wilcoxon signed-ranks test [60] to test the five null hypotheses statements which were stated as follows: "*H0.i The median of the construct Ci is <= 5.0*", where *i* goes from 1 to 5. We expected to reject all of the five null hypotheses with an alpha error of 0.05 as a maximum, and with it to obtain initial evidence on satisfactory perceptions for the SOCA methodology on usefulness, ease of use, compatibility, result demonstrability, and final behavioral intention of use in the pilot group of 32 evaluators. We considered that by using a Likert scale from 1 to 7, the fixed test value of 5.0 implies at least a moderately good score. For this task, we used the free statistical tool MaxStatLite (www.maxstatlite.com). Table 6 reports the associated statistical results.

Thus, in case the null hypotheses are rejected we obtain that our SOCA methodology fulfills the five metrics (i.e. it is perceived as useful, ease of use, compatible, with result demonstrability, and with a final behavioral intention to use in the near future) as the pilot group of evaluators (expert panel) previously suggested.

In this empirical statistical evaluation, we did not pursue to elaborate or test a predictive theory with a predicted construct of behavioral intension of use. Although this can be elaborated with specific statistical techniques like PLS [20], this is left for future research.

Table 6. Results of the Empirical Evaluation of the SOCA Development Systems Engineering Methodology with a Pilot Group of 32 Part-Time MSc Students

Null Hypothesis	Test Median	32 Sample Median	Test Statistics	Alpha value	P-value	Reject Null Hypothesis H0?
H0.1 "The median of the construct **usefulness** is less or equal to 5.00"	**4.000**	**5.625**	**63.000**	**0.0500**	**< 0.0001**	**YES**
H0.2 "The median of the construct **ease of use** is less or equal to 5.00"	**4.000**	**5.667**	**21.00**	**0.0500**	**< 0.0001**	**YES**
H0.3 "The median of the construct **compatibility** is less or equal to 5.00"	**4.000**	**4.833**	**94.500**	**0.0500**	**0.0038**	**YES**
H0.4 "The median of the construct **result demonstrability** is less or equal to 5.00"	**4.000**	**5.000**	**30.500**	**0.0500**	**< 0.0001**	**YES**
H0.5 "The median of the construct **behavioral intention of use** is less or equal to 5.00"	**4.000**	**5.250**	**22.00**	**0.0500**	**< 0.0001**	**YES**

5.3 Discussion

SOCA development methodologies are relatively new. Several SOCA methodologies have been reported in the last decade [25], but none of them has gained sufficient acceptance for SOCA academic and professional community, as it happened in the previous OOSE and CBSE paradigms. In [25] it is presented several generic process-related properties of SOCA development methodologies that allows us to compare SOCA engineering approaches. In particular, the objective/scope of the methodology is a relevant feature, and they distinguished five types: 1) focused on a full development process, 2) focused only on analysis and design, 3) focused only on service composition, 4) focused only on migration of SOA, and 5) focused only on project management. This categorization matches with different levels of abstraction of the methodology, from the management project, passing through the architecture perspective and process engineering, to the service composition. Our SOCA methodology is type 1.

SOCA development studies appear because of the novelty of the SOSE paradigm and the need of methodologies for service-oriented systems development. Although there are new proposals for SOCA development, they have several limitations. A desirable issue to be addressed by such methodologies is the ability to narrow the conceptual gap between the problem (or business) domain and the system implementation (or application) domain [18][61]. However, current proposals do not tackle this issue since they do not consider the analysis and design of services, composition, orchestration, and choreography all together. Furthermore, in general terms, these methodologies do not cover all the software development life-cycle [25]. In contrast, our SOCA methodology fills this gap by (i) including the product "Enterprise Architecture design" -although limited to business vocabulary-; (ii) considering some activities for planning, (iii) aligning Business with Information Technology since the beginning to the end of the development process -e.g. requirements analysis, a service oriented analysis and design, at architectural and system and the user interface design, and an operative evaluation at the business level-, (iv) suggesting activities to ensure an operative deployment at the architecture level; and (v) evaluating the operative system at the application level for correctness and for system evolution [56]. Also, our proposed methodology covers all the software system development life-cycle in a well-structured manner i.e. our

methodology includes phases, activities, models, and products as well as roles and an iterative development process.

The results of the empirical evaluation have shown that our methodology is practical and applicable for developing SOCAs. Specifically, the first evaluation involving a 5-point Likert scale was applied to an expert panel and obtained 4.50 or higher for each evaluated item, an average of 4.66, and an average standard deviation of 0.62. Such results suggest that the panel of experts on software engineering methodologies considered our SOCA methodology as having strong theoretical support (item P1) while considering that our proposal covers state of the art principles (item P2) without important omissions (item P3). The pilot group also considered our methodology as logically coherent (item P4) with a suitable conceptual model (item P5) as well as considering it as a real contribution that is not a duplicate of an existent solution (item P6). The second evaluation involving a 7-point Likert scale was applied to a group of software engineering practitioners and obtained a value higher than 4.0 for all the evaluated constructs with the directional hypothesis stating "H0.i The median of the construct C*i* is <= 4.0", where *i* goes from 1 to 5. We can interpret this result as all constructs were perceived as no negative or neutral ones. Reported reliability [31] for the constructs measured is higher than 0.70, which is the expected value for pilot studies. The results reported in Table 6, and supported by the Wilcoxon signed-ranks test, provide statistical-based evidence that SOCA methodology was perceived as useful, easy to use, compatible, with result demonstrability, and with a final intention to use it. Finally, Table 7 presents a comparative analysis of previous service oriented development methodologies with our work. (✓ indicates when the methodology gives guidance or defines activities related to the topic of the development processes/areas, while a blank square indicates no guidance is given for the topic). Although, all compared methodologies support at least one of three topics, namely Business Process Modeling, Requirements Engineering or Service Requirements, none of the revised methodologies provides a complete solution for all topics. However, our methodology covers all topics except requirements engineering and automated transformation.

Table 7. Comparative analysis of the SOCA Development Systems Engineering Methodology with other methodologies

Topic / area of the development processes.	M1 (Karastoyanova, 2003) [33]	M2 (Kotonya, 2004) [39]	M3 (Ivanyukovich, 2005) [28]	M4 (Kühne, 2005) [40]	M5 (Cox, 2005) [14]	M6 (Karakostas, 2006) [32]	M7 (Papazoglou, 2007) [51]	M8 (Gu, 2009) [23]	M9 (Cantor, 2003) [10]	M10 (Arsanjani, 2008) [3]	M11 SOCA Dev. Systems Eng. Methodology
Requirements Engineering		✓		✓				✓			
Business Process Modeling			✓		✓		✓	✓	✓	✓	✓
Enterprise architecture design										✓	✓
Service requirements and Service Design			✓			✓	✓		✓	✓	✓
Subsystems architectural design		✓			✓					✓	✓
SOA Design (service composition and orchestration)						✓	✓			✓	✓
Application modelling	✓		✓					✓	✓		✓
Automated transformation	✓			✓		✓					
Monitoring & Maintenance		✓			✓		✓	✓		✓	✓
Propose products / artifacts									✓	✓	✓
Propose to elaborate database									✓		✓

There are three methodologies whose focus is on the automatic transformation towards a set of services and compositions of executable services, specifically those proposed by Karakostas, Kühne and Kotonya's focuse mainly on requirements engineering, architecture design and monitoring and maintenance.

Unlike our methodology -except Karastoyanova, Ivanyukovich, Gu and Cantor (RUP-SE)-, the compared approaches do not support the topics of SOA design (i.e. composition and orchestration of the services). Therefore, current approaches do not have neither an implicit nor explicit vision of SOCA. In contrast, our methodology emphasizes -as part of the modeling of the application- the importance of the design of the user interface, and business alignment throughout the development process. Also, our methodology proposes both activities and products/artifacts to be built.

5.4 Limitations and Threats to Validity

There are a number of threats to validity regarding the statistical methods we employed. According to [63][67], when we use a survey research method, it must be considered the following validation issues: 1) *instrument validation*, 2) *internal validity*, 3) *statistical conclusion validity*, and 4) *external validity*. *Instrument validation* refers to using a reliable instrument that contains items (operational variables) strongly associated with the expected construct to be measured. The *instrument validation* must satisfy a *content validity*, a *construct validity*, and a *reliability of constructs*. We addressed threats to this type of validity by using well-tested instruments to measure the constructs of usefulness, ease of use, compatibility, result demonstrability and behavioral intention of use [31]. In [31] *content validity* was assessed by reusing a previous valid instrument. *Construct validity* was assessed through a satisfactory matrix of loadings factors for each construct (see pp. 210-211), where each value is greater than 0.70 and its maximum value belongs only to an associated construct. The *reliability of constructs* was assessed with the test of Cronbach's Alpha [31] and four of the five values were satisfactory obtaining higher or equal than 0.80 whereas only the construct Result Demonstrability exhibited a score of 0.70. Thus, the supported hypothesis H0.4 must be considered with caution.

Internal validity refers specifically to the extent to which an experimental variable is responsible for any variance in a dependent variable. This is also conceptualized as the reduction of alternative/competitive interpretations on the obtained results. In this research, it involves the fact that the pilot sample of evaluators reported satisfactory scores for the 5 constructs for other reasons different from the direct evaluation of our SOCA methodology. These scores can be biased for instance when there is pressure on the evaluators to assess with high scores, empathy of the evaluators with the research team, or lack of interest of the evaluators. The main threats for *internal validity* are: *history, maturation, testing, instrumentation, mortality,* and *selection* [67]. *History* refers to bias on scores when a critical event happened before conducting the evaluations. *Maturation* refers to a bias caused by time-passing effects on evaluators. *Testing* refers to bias on evaluators when they had previously evaluated the methodology. *Instrumentation* refers to changes with the instruments used. *Mortality* refers to the case when an evaluator dies before filling out all the questionnaires of the survey. Lastly, *selection* refers to the selection of evaluators to participate in one of two groups -experimental or control group-. The collection of data in this research avoids

the threats of *history* because no critical external event was registered during the evaluation process. *Maturation* was avoided because evaluations were transversal (no longitudinal) ones. *Testing* was avoided because a previous evaluation was not conducted. *Instrumentation* was avoided because we used the same valid instrument. *Mortality* was avoided because evaluators participated only once. Lastly, s*election* is only employed when both an experimental and a control group take place in the survey, this, in order to decide, which elements conform to each group.

Statistical conclusion validity refers to the correct utilization of statistical procedures. In this research, we avoided threats with the utilization of the adequate statistical test (a one-tailed Wilcoxon signed-ranks Test [60]) and by using an adequate alpha value (Type I error) of 0.05. Also, *external validity* refers to the extent to which the results can be generalized to other populations, settings or contexts. In this research, despite the pilot sample of evaluators is highly representative of the population, which is generically defined as "*population of software engineering practitioners working in medium and large enterprises in developing economies*", we cannot claim these results can be generalized to other similar populations because the pilot sample of evaluators were selected by a non-probabilistic procedure. Thus, the *external validity* must be limited to this type of sample.

Finally, given that small development teams and a small project were employed for evaluating this research, our claims must be considered as initial insights toward more definitive results. However, we believe our preliminary results are encouraging.

6 Conclusion

We have proposed a Service-oriented Computing Application Development Systems Engineering Methodology which was derived by employing four core design building blocks, namely Service-oriented Computing, the Model-Driven Architecture, two popular development methodologies -RUP-SE and MBASE-, and agility-rigor balance recommendations. Our methodology extends the existent and emergent body of knowledge in Service-Oriented Software Engineering by applying and enhancing the existent knowledge. Our methodology also includes features from the main SOCA development proposals and proposes the application of emergent techniques that are useful for SOCA development.

From a practitioner perspective, this research contributes with the following results. First, we defined a new agility-rigor balanced methodology. Second, we proposed a development methodology that provides a set of new techniques that are more suitable for SOCA development than those used in Object-Oriented Software Engineering or Component-Based Software Engineering approaches. Lastly, we used a running example that encourages practitioners to use and test our methodology.

Finally, we consider this study raises new interesting issues that require further research. Firstly, evaluating our methodology with empirical case studies involving a variety of business organizational problems will make it possible to capture more definitive quantitative usability metrics such as usefulness, ease of use, compatibility, and value of using our methodology for developing SOCAs. Secondly, experiments can be conducted with graduate part-time students to contrast the usability metrics obtained by using the most known software development methodologies such as RUP, MSF, and

RUP for SOA, versus our approach. Thirdly, an agile version of our methodology can be elaborated as well as identifying for what kinds of software projects are recommended the full methodology and for what kinds of projects can be suitable the agile version. Lastly, tool support can be developed for automating the model-to-model transformations.

References

1. Allen, E, Seaman, C.: Likert Scales and Data analyses, Quality Progress, Statistics Roundtable, 40(7), 64-65. (2007)
2. Amsden, J.: Modeling SOA IBM (parts 1,2,3,4,5), Level: Introductory, STSM, IBM, 2007
3. Arsanjani, J., Hailpern, B., Martin, J., Tarr, P.L.: Web Services: Promises and compromises, IBM Research Division, Thomas J. Watson Research Center, 1-18. (2008)
4. Avison, D., Fitzgerald, G.: Where now for development methodologies?, Communications of the ACM, 46(1), 78-82. (2003)
5. Baghdadi, Y.: A comparison framework for service-oriented software engineering approaches: Issues and solutions, International Journal of Web Information Systems, 9(4), 279-316. (2013)
6. Beisiegel, M., Blohm, H., Booz, D., Edwards, M., Hurley, O., et al.: SCA Service Component Architecture: Assembly Model Specification, SCA Version 1.00. (2007)
7. Bianco, P., Kotermanski, R., Merson, P.: Evaluating a Service-Oriented Architecture, Software Engineering Institute, Technical Report, 1-89. (2007)
8. Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., Madachy, R.: Using the WinWin spiral model: a case study", IEEE Computer, 31(7), 33-44. (1998)
9. Boehm B. and R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed, Addison Wesley, Boston, (2004).
10. Cantor, M.: Rational Unified Process for Systems Engineering, Rational Brand Services IBM Software Group. (2003)
11. Carter, L., Belanger, F.: The Influence of Perceived Characteristics of Innovating on e-Government Adoption, Electronic Journal of e-Government 2(1), 11-20. (2004)
12. Center for Software Engineering: Guidelines for Model-Based (System) Architecting and Software Engineering (MBASE), University of Southern California. (2000)
13. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stanfford, J.: Documenting Software Architectures: Views and Beyond, 2nd Edition, Pearson, Addison-Wesley. (2011)
14. Cox, D.E., Kreger H.: Management of the service oriented-architecture life-cycle, IMB Systems Journal, 44(4), IBM Corporation. (2005)
15. Dix, A., Finlay, J., Abowd, G.D., Beale, R.: Human-Computer Interaction, 3rd Edition, Pearson, Prentice-Hall. (2004)
16. Dumas, M.: Towards a semantic framework for service description, Cooperative Information Systems Research Centre, Queensland University of Technology, Australia. (2000)
17. Ellis, B., Stylos, J., Myers, B.: The factory pattern in API design: A usability evaluation, In Proc. of the 29th international conf. on Soft. Eng., IEEE Computer Society, 302-312. (2007).
18. France, R., Rumpe, B.: Model driven development of Complex Software: A Research Roadmap, Department of Computer Science Colorado State University. (2007)
19. Garriga, M., Mateos, C., Flores, A., Cechich, A., Zunino, A.: RESTful service composition at a glance: A survey, Elsevier, Journal of Network and Computer Applications, 60(1) 32-53. (2016)
20. Gefen, D., Straub, D., Boudreau, M.: Structural equation modeling and regression: Guidelines for research practice, Communications of the Association for Information Systems, 4(1), 1-79. (2000)

21. Gefen, D., Karahanna, E., Straub, D.: Trust and TAM in online shopping: an integrated model, MIS Quarterly 27(1), 51-90. (2003)
22. Granollers, T., Lorés, J. & Perdrix, F.: Usability Engineering Process Model. Integration with Software Engineering. In Proceedings of HCI International 2003, 1-6. (2003)
23. Gu, Q., Lago, P.: A stakeholder-driven service life cycle model for SOA, ACM IW-SOSWE'07, Dubrovnik, Croatia. (2009)
24. Gu Q., Lago P.: Service Identification Methods: A Systematic Literature Review, In: Di Nitto E., Yahyapour R. (eds) Towards a Service-Based Internet. Service Wave, Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, 6481, 37-50. (2010)
25. Gu, Q., Lago, P.: Guiding the selection of service-oriented software engineering methodologies, Springer, SOCA 2011(5), 203-223. (2011)
26. Hassanzadeh, A., Namdarian, L.: Developing a framework for evaluating service oriented architecture governance (SOAG), Knowledge-Based Systems 24(5), 716-730. (2011)
27. Hevner, A.R.: Design Science in Information Systems Research, Information Systems and Decision Sciences, MIS Quarterly 28(1), 75-15. (2004)
28. Ivanyucovich A., Gandaharan, G.R., D'Andrea, V., Marchese, M.: Toward a service-oriented development methodology, University of Trento, Italy. (2005)
29. Jørgensen, M., Jøberg, D. I.: Impact of effort estimates on software project work, Information and software technology, 43(15), 939-948. (2001)
30. Kaba, B: Modeling information and communication technology use continuance behavior: Are there differences between users on basis of their status?, International Journal of Information Management, 38(1), 77–85. (2018)
31. Karahanna, E., Straub, D. W., Chervany, N. L.: Information Technology adoption across time: A cross-sectional comparison of pre-adoption and post-adoption beliefs, MIS Quarterly, 23(2), 183-213. (1999)
32. Karakostas, B., Zorgios, Y., Alevizos, C.C.: Automatic derivation of BPEL4WS from IDEF process models, Software & System Modeling, 5(2), 208-218. (2006)
33. Karastoyanova, D.: A methodology for development of Web Services-based Business Processes, Thechnische Universität Darmastadt. (2003)
34. Keith, M., Demirkan, H., Goul, M.: Service-oriented methodology for systems development, Journal of Management Information Systems, 30(1), 227-260. (2013)
35. Kitchenham, B., Linkman, S., Law, D.: Critical review of quantitative assessment. Software Engineering Journal, 9(2), 43-54. (1994)
36. Kiwanuka, A.: Acceptance Process: The Missing Link between UTAUT and Diffusion of Innovation Theory, American Journal of Information Systems 3(2), 40-44. (2015)
37. Kontogiannis, K., Lewis, G. A., Smith, D. B., Litoiu, M., Muller, H., Schuster, S., Stroulia, E.: The landscape of service-oriented systems: A research perspective, In Proceedings of the International Workshop on Systems Development in SOA Environments, IEEE Computer Society. (2007)
38. Kosar, T., Mernik, M., Carver, J. C.: Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments, Empirical software engineering, 17(3), 276-304. (2012)
39. Kotonya, G., Hutchinson, J., Bloin, B.: A method for formulating and architecting component and Service-oriented Systems, Computing Dept., Lancaster University. (2004)
40. Kühne, S., Thränert, M., Speck, A.: Towards a methodology for orchestration and validation of cooperative e-business components, Inst. of Cybernetics, Tallinn Technical Univ. (2005)
41. Legris, P., Inghamb, J., Collerettec, P.: Why do people use information technology? A critical review of the technology acceptance model, Information & Management 40(3), 191–204. (2003)
42. Miller, J., Mukerji, J.: MDA guide version 1.0.1, Object Management Group (OMG), (2003)
43. Mooi, E., Sarstedt, M.: Concise Guide to Market Research: The Process, Data, and Methods Using IBM SPSS Statistics, Springer, Germany (2018)

44. Mora, M., Gelman, O., Paradice, D., Cervantes, F.: The case for conceptual research in information systems, In Proceedings CONF-IRM 2008, p. 52. (2008)
45. Mora, M., Rory, V., Rainsinghani, M., Gelman, O.: Impacts of electronic process guides by types of user: An experimental study, International Journal of Information Management, 36(1), 73-88. (2016)
46. Morris, M. G., Dillon, A.: How user perceptions influence software use, IEEE software, 14(4), 58-65. (1997)
47. Okoli, C., Pawlowski, S. D.: The Delphi method as a research tool: an example, design considerations and applications, Information & management 42(1), 15-29. (2004)
48. Oktaba, H., Ibargüengoitia, G.: Software process modeled with objects: Static view, Computación y Sistemas 1(4), 228-238. CIC-IPN ISSN 1405-5546. (1998)
49. OMG: OMG Meta Object Facility (MOF) Core Specification, v2.5.1. (2016) [Online]. Available: http://www.omg.org/spec/MOF (Current 2017)
50. Papazoglou, M.P., Traverso, P., Schahram, D., Leymann, F., Bernd, J.: Service-oriented Computing: Research roadmap, Tilburg University, The Netherlands. (2006)
51. Papazoglou, M.P., Van Den Heuvel W.: Business process development life cycle methodology. Communications of the ACM, 50(10), 79-85. (2007)
52. Papazoglou, M.P., Van Den Heuvel, W.J.: Service oriented architectures: approaches, technologies and research issues, The VLDB Journal, 16(3), 389-415. (2007)
53. Riemenschneider, C., Hardgrave, B., Davis, F.: Explaining software developer acceptance of methodologies: a comparison of five theoretical models, IEEE transactions on Software Engineering, 28(12), 1135-1145. (2002)
54. Rodriguez da Silva, A.: Model-driven engineering: A survey supported by the unified conceptual model, Computer Languages, Systems & Structures 43 (2015) 139–155. (2015)
55. Rodriguez, L.C., Mora, M., Alvarez, F.J.: Process Models of SDLCs: Comparison and evolution, Handbook of Research on Modern Systems Analysis and Design Technologies and Applications 76-89, Minnesota State University, Mankato, USA, IGI Global. (2008)
56. Rodriguez-Martinez, L.C., Mora, M., Alvarez, F., Garza, L.A., Duran-Limon, H.A., Muñoz J.: Review of Relevant System Development Life Cycle (SDLC) in Service-oriented Software Engineering (SOSE), Journal of Applied Research and Technology, 10(2), 94-113. (2012)
57. Rodriguez-Martinez, L.C.: A Case Example for Illustrating SOCA Development Systems Engineering Methodology and Evaluation Questionnaires: Complementary Documents. (2018). http://hduran.cucea.udg.mx/publications/SOCA-paper-extra-resources.zip.
58. Sargent, R.: Validation and verification of simulation models, Proceedings of the 1999 Winter Simulation Conference, 39-48. (1999)
59. Sheng, Q.Z., Qiao, X., Vasilakos, A.V. Szabo, C. Bourne S., Xu, X.: Web services composition: A decade's overview, Elsevier, Information Sciences, 280(1), 218-238. (2014)
60. Sheskin, D.: Handbook of parametric and nonparametric statistical procedures, CRC Press. (2003)
61. Sigh-Walia, G., Carver, J.C.: A systematic literature review to identify and classify software requirement errors, Elsevier, Information and Software Technology. (2009)
62. Sommerville, I.: Software Engineering, 7th Edition, Pearson, Addison-Wesley. (2005)
63. Straub, D.: Validating instruments in MIS research. MIS quarterly, 147-169. (1989)
64. Van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour, In Proc. of the Fifth IEEE International Symposium on Requirements Engineering, 249-262. (2001)
65. Vavpotic, D., Bajec, M.: An approach for concurrent evaluation of technical and social aspects of software development methodologies, Information and software Technology, 51(2), 528-545. (2009)
66. Venkatesh, V., Morris, M., Davis, G., Davis, F.: User Acceptance of Information Technology: Toward a Unified View, MIS Quarterly 27(3), 425-478. (2003)
67. Wohlin, C., Höst, M., Henningsson, K.: Empirical research methods in software engineering. In Empirical methods and studies in software engineering, Springer, 7-23. (2003)

68. Zikmund, W., Babin, B., Carr, J., Griffin, M.: Business research methods, Cengage Learning. (2013)
69. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service Oriented Analysis and Design: an approach of interdisciplinary modeling for SOA projects, IBM Developer Works. (2004) [Online]. Available: https://www.ibm.com/developerworks/library/ws-soad1/index.html (Current 2017)

Laura C. Rodriguez-Martinez is a full Professor at the Systems and Computing Department, Institute of Technology of Aguascalientes, Mexico. She holds a PhD in Computer Science at Universidad Autonomous University of Aguascalientes, Mexico in 2009. Her research interests include Software Systems Development Processes, Service-Oriented Software Engineering and Graphical-User Interfaces Development Processes.

Hector A. Duran-Limon is currently a full Professor at the Information Systems Department, University of Guadalajara, Mexico. He completed a PhD at Lancaster University, England in 2002. His research interests include Cloud Computing and High Performance Computing (HPC). He is also interested in Software Architectures, Software Product Lines and Component-based Development.

Manuel Mora is a full-time Professor at the Autonomous University of Aguascalientes (UAA), Mexico. Dr. Mora holds an Eng.D. in Engineering (2003) from the National Autonomous University of Mexico (UNAM). He has published over 90 research papers in international conferences, research books, and journals listed in the JCR. Dr. Mora is a senior member of ACM (since 2008) and of the Mexican National Research System at Level I, and does research focused on Design Methodologies for IT Services Systems and Decision-Making Support Systems (DMSS).

Francisco Alvarez Rodríguez is a Professor of Software Engineering. He is PhD from the National Autonomous University of Mexico. He has published research papers in several international conferences in SwE and e-Learning process. His research interests are SwE lifecycles for small and medium sized enterprises and SwE process for e-learning. He is currently president of the National Council for Accreditation of programs and Computing, A.C. (CONAIC).

Received: July 3, 2017; Accepted: September 20, 2018