APPROVAL SHEET

Title of Dissertation: Resuscitating Service Oriented Architecture (SOA): Honing SOA Adoption by Measuring Maturity at the Service Architecture Level

Name of Candidate: Gohar Mukhtar
Ph.D. Information Systems, 2016

Dissertation and Abstract Approved: 

Anthony F. Norcio, PhD
Professor
Information Systems

Date Approved: 3/31/2016

ABSTRACT

Title of Document: RESUSCITATING SERVICE ORIENTED ARCHITECTURE (SOA): HONING SOA ADOPTION BY MEASURING MATURITY AT THE SERVICE ARCHITECTURE LEVEL.

Gohar Mukhtar, Ph.D. Information Systems, 2016

Directed By: Professor Anthony F. Norcio Name. Information Systems

Despite sound theory, Service Oriented Architecture (SOA) appears to be failing in delivering on its brilliant promises. Over the last few years, a vast amount of good-faith efforts by many organizations towards SOA adoption have withered in frustration. Researchers and practitioners cannot pinpoint the reason(s) with certainty. Some point to its inherent complexity while others point to the widespread misunderstanding and confusion this over-used buzzword has acquired in the industry. One thing everyone agrees is that SOA adoption is non-trivial. The magnitude of change required to transition from traditional silo-based application design to service-orientation can be overwhelming for any organization. This architectural paradigm-shift encapsulates more than just new technologies; it demands a different mindset. The fundamental service design principles, however, remain applicable throughout. And this is where the others have missed the heart of the problem. The SOA adoption maturity is commonly measured at a high-level, involving elements like infrastructure, tools and technologies adoption, but misses to consider the low-level service design principles. This study offers a new focused maturity-measure that is capable of assessing the essence of SOA design paradigm at its roots. The proposed *Service Architecture Maturity* (SAM) not only provides a more accurate measure of SOA adoption, but can also hone in on the adoption efforts by focusing attention to the

essentials. SAM can help organizations currently stalling on the path of SOA adoption in recognizing their misplaced focus, correcting mistakes, and in achieving the promised benefits from SOA adoption.

RESUSCITATING SERVICE ORIENTED ARCHITECTURE (SOA): HONING SOA ADOPTION BY MEASURING MATURITY AT THE SERVICE ARCHITECTURE LEVEL

By

Gohar Mukhtar

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy,
Information Systems
2016

Dedication

To my ever beautiful and radiant wife Nazli who stood by me through all the ups and downs of life with unwavering commitment, love and loyalty. She indeed has a heart of gold, and I'm the lucky one to have it.

Acknowledgements

I am heartily thankful to my research supervisor, Prof. Anthony F. Norcio, whose encouragement, guidance and support from the beginning to the end enabled me to complete this profound academic journey. His commitment to excellence and high academic standards made it a great learning experience. I can't thank him enough for all his help and support.

I want to express my gratitude to my doctoral committee members, Dr. Lina Zhou and Dr. Henry H. Emurian for their constructive comments that improved the quality of this dissertation. I am especially grateful to Dr. Gregory Bluher and to Dr. Massoud Farhang for being the designated readers of this work.

I would also like to thank my employer, IRS, Terence V. Milholland (CTO) and Daniel B. Chaddock (ACIO) for their trust, confidence and continued support during this research and pursuit of my Ph.D. at UMBC.

This acknowledgement will not be complete without thanking my wife Nazli and my daughters Zernab and Amal for enduring lonely evenings and weekends for the past six years while I was preoccupied with my academic activities.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1. SOA Overview

Ongoing empirical studies, following the up-and-coming trends in the Information Technology (IT) industry, show strong support for Service Oriented Architecture (SOA)[1] adoption. In 2008, at least 44% of North American, European, and Asian-Pacific enterprises adopted SOA, and at least 63% would adopt it by the end of 2008, with the trend only growing stronger in more recent years (Aldris, Nugroho, Lago, & Visser, 2013). Resent research shows the value and applicability of SOA in such emerging technologies and diverse areas like design and development of mobile applications (Ali, Chen, & Solis, 2012), air traffic management (Gringinger, Trausmuth, Balaban, Jahn, & Milchrahm, 2012), industrial automation (Ollinger, Zuhlke, Theorin, & Johnsson, 2013), systems of systems (Lewis, Morris, Simanta, & Smith, Jan-Feb 2011), next generation network architecture and virtualization in cloud computing (Duan, Yan, & Vasilakos, 2012; Grammatikou, et al., 2011; Kumar, Haber, Yazidi, & Reichert, 2010; Ning, Zhou, Zhang, Yin, & Ni, 2011; Ruz, Baude, Sauvan, Mos, & Boulze, 2011), generic virtual power plants and smart power grid management (Andersen, Poulsen, Trholt, & Ostergaard, 2009; Vrba, et al., Aug 2014) smart building management systems (Degeler, et al., 2013), telecommunications (Blum, Magedanz, Schreiner, & Wahle, 2009), driver assistance systems (Wagner, Zobel, & Meroth, 2014), and in business, financial services industry (Fischbach, Puschmann, & Alt, 2011; Murer & Hagen, Nov-Dec 2014),

[1] Service-oriented architecture is an architectural style for building service-oriented solutions with distinct characteristics in support of realizing service-orientation and the strategic goals associated with service-oriented computing (for more information, see the Glossary of terms at the end of this document).

healthcare (Chu, 2005) and military strategic and tactical networks/systems (Cameron, Stumptner, Nandagopal, Mayer, & Mansell, 2013; Lund, Eggen, Hadzic, Hafsoe, & Johnsen, October 2007; Nath, 2012). Prior research has also shown that with the adoption of SOA, changeability and efficiency improve compared to alternative architectures (Offermann, Hoffmann, & Bub, 2009).

Despite all this visibility and traction gained by SOA during the last few years in the IT industry, it remains wrapped in significant mist of confusion. The conflicting vendor interests and technology hypes have contributed in perpetuating this widespread bewilderment (Bloomberg, 2013). The misperception ranges from some considering the use of web services technology as constituting SOA, to others who deem SOA as a web-centric mutation of the existing Object Oriented design style (Bloomberg, 2013). Some researchers have gone so far as to assert "that one of the most important challenges that face SOA is the lack of knowledge" and understanding of its finer aspects (Hassan, 2009). In what follows, some salient features of SOA are reviewed in order to demystify the subject matter.

SOA is a unique architectural style that has gained significant attention within the information technology and business communities (Service Oriented Architecture Reference Model Technical Committee, 2012). It represents a new generation of distributed architecture (Li & Wu, 2009) in which, through meaningful "application of service-orientation design principles, *Service-oriented solution logic* is shaped" (Erl, 2008, pp. 38-39) and exposed through standardized service contracts, specifically designed to communicate via non-proprietary protocols and technologies, based on established industry standards (Curbera, 2007). To be legitimately considered an SOA, "a

system should demonstrate some degree of *service-orientation*" (Aldris, Nugroho, Lago, & Visser, 2013), where "S*ervice-orientation* is a design paradigm comprised of *service-orientation design principle*" (Erl, 2008, p. 41). SOA builds upon past distributed computing efforts, adds new design and governance considerations, and continues to evolve in face of changing research challenges (Erl, 2008, pp. 96-97; Takdir & Kistijantoro, 24-25 Sept. 2014).

The concept of a *Service* sits at the heart of this design paradigm as an atomic, *Cohesive*, and black-boxed unit of service-oriented solution-logic (Hassan, 2009; Fortuna & Mohorcic, Aug 2009). It is defined as a self-contained logical representation of a repeatable business activity that has a specified outcome (The Open Group, 2009). An organization involved in service-orientation usually strives to build a collection of services – known as a *Service Inventory* – that are designed, developed, deployed and governed according to one set of standards (Figure 1). It is preferable for an enterprise to have one service inventory; however, multiple service inventories within very large enterprises that are comprised of culturally independent business domains are also not uncommon (Mauro, Leimeister, & Krcmar, 2010).

Figure 1: High-level view – a *Service Inventory* is a pool of services that are designed, build, deployed and governed according to the same set of standards.

SOA as an architectural style focuses on optimizing creative aggregations of these services for dynamically solving business-process automation problems. Each service is assigned a distinct *functional-context*, which is comprised of a set of *service capabilities* related to this context (Erl, 2008). Thus, a service can be conceived as a set or container of related functions, and the technology-neutral term for those related functions is service capabilities (Erl, 2008, p. 115). Based on business perspective, a functional-context is a meaningful affinity which binds related capabilities into services, just like the *Cohesion* which binds methods into classes in object-oriented design paradigm. These discrete capabilities are invoked by consumer programs and/or other services through a standardized and published service contract.

Based on the theory of *separation of concerns*, services can be modeled after three[2] basic types, i.e. *Task*, *Entity* or *Utility* (Erl, 2008, p. 43). Task services are also sometimes called Process Services (or Business Process Services) because they are business process-

[2] Some experts add *Orchestrated-Task* as a fourth type, but it can also be looked at as a variant of basic *Task* type. Process Abstraction and Process Centralization are the two design patterns that are applied to a Task service to make it an Orchestrated-Task service.

centric in nature, and their functional-context typically encapsulates a business-workflow. Entity services are not process-centric but are business-centric. This means that they are agnostic to any specific business-process but hold affinity to specific business domains. Utility services are most generic in nature, and thus are completely agnostic; they are neither process-centric nor business-centric, and are thus most reusable (Figure 2).

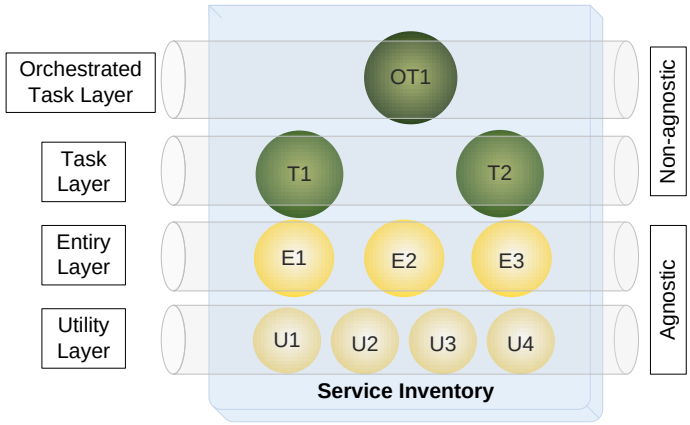


Figure 2: A deeper look into the layered structure of a Service Inventory – a low-level view.

Although the individual services exist as physically independent software programs with specific design characteristics in support of attaining the strategic goals associated with service-oriented computing, it's really the aggregation of these services – called a *Service Composition* – that typically solves a complete business problem.

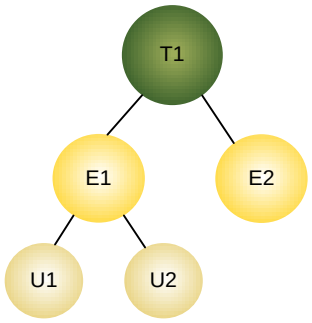


Figure 3: A sample Service Composition using all three types of services. A Task service (T1) encapsulates business process work-flow logic; an Entity service (E1, E2) encapsulate business-centric but process-agnostic logic; and Utility services encapsulate generic and completely agnostic logic.

Service Compositions are usually aggregated from within a Service Inventory to automate a business-process. In such cases, a Task service, encapsulating a business-workflow, acts as a *Composition Controller* (T1 in Figure 3 and Figure 4) and does the composition (or aggregation) of the Entity and Utility type services. This kind of service-modeling is based on *the theory of separation of concerns* which fosters modularity, and promotes reutilization, understandability, extensibility and maintainability by separating the different concerns of an application into independent modules (Felix & Ortin, 2014). Thus, large business problems or processes are decomposed into smaller components to further their reusability with new and creative service compositions (Dubey, 2010).

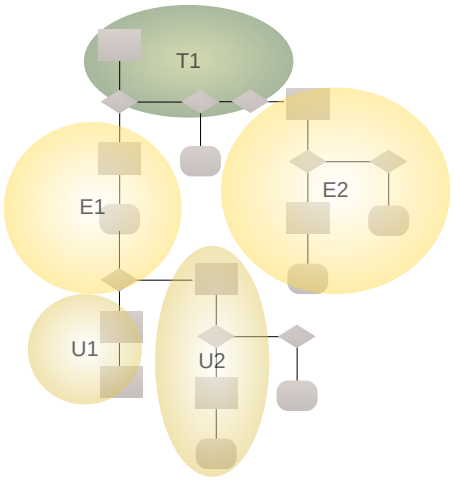


Figure 4: A big business problem is decomposed into smaller modules. This modularization approach allows for proper service modeling. Each type of service model encapsulates only appropriate kind of logic. This typically results in a Composition of all three types of services.

Central to this architectural style is a well-recognized and established set of *eight principles of service design* (Erl, 2008). These principles provide specific guidelines about how to shape the individual services that are to become part of the larger whole called an *SOA Ecosystem*. From an architectural perspective, these design principles are the cornerstone of an SOA adoption initiative in any organization. However, due to vendor-interest and technology-hypes, these principles are sometimes relegated to a backseat in the SOA adoption initiatives (Bloomberg, 2013). When this happens, an organization often loses focus from the essentials, gets confused by mere tools and technologies, and gets "deluded into thinking that they're building SOA, when actually they aren't doing any such thing. Such situations are still quite prevalent, and when such implementations fail, they can undeservedly give SOA a bad name". (Bloomberg, 2013, pp. 23, 180)

2. ***Promised Benefits of SOA Adoption***

What are the core issues that SOA architectural style attempts to solves, and why should an organization care to invest in SOA adoption?

It is commonly understood that "SOA can reduce integration cost, increase visibility and agility, increase asset reuse, and ease regulatory compliance" (Bloomberg, 2013, p. 29; Sud, 2010). Through industry experience, a set of standard "strategic goals and benefits" have emerged from the SOA vision, as illustrated by the Figure 5 (Erl, 2008, p. 56). These goals and benefits clearly establish a target-state that no IT enterprise can ignore and stay competitive (Jain & Kumar, 2007).

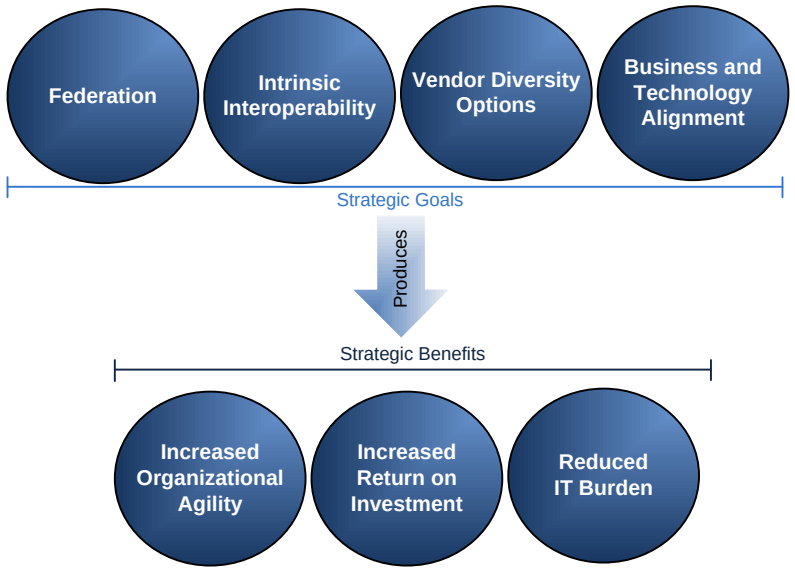


Figure 5: SOA Goals and Benefits. The first four goals lead to the attainment of the later three goals and benefits. All SOA goals are interrelated and are strategic in nature: they are focused on the long-term benefits of an IT enterprise.

a. Intrinsic Interoperability

b. Vendor Diversity Options

c. Federation

d. Business and Technology Alignment

e. Organizational Agility

f. Increased Return On Investment (ROI)

g. Reduced IT Burden

2.1. Intrinsic Interoperability

Interoperability means the ability of the disparate software programs to work together by sharing and exchanging data. “Intrinsic interoperability” is so inherent and fundamental to the design of such disparate programs that no (or minimal) amount of integration is required for them to work together as a whole (Erl, 2008, pp. 92-93). In other words, integration can be looked at as being antithesis of interoperability; more integration two programs require to interact, less interoperable they are. The integration effort is often costly (Murer & Hagen, Nov-Dec 2014), and can grow exponentially as the complexity and size of the programs increases (Appendix F).

Not only that SOA sets intrinsic interoperability as one of its goal, increased “interoperability is a natural by-product of applying service-orientation design principles” (Erl, 2008, p. 75). By being mindful of this goal at the design-time of these disparate programs called Services, SOA seeks to reach a point where integration as a concept starts to fade away; the target-state being an inventory of services that can work in Service Compositions inherently and without any special integration effort (Figure 6).

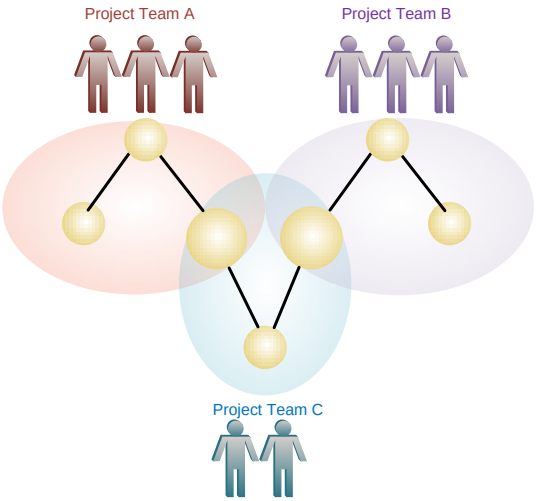


Figure 6: Interoperability. Because these services are designed to be intrinsically interoperable, regardless of when, where and by whom they are developed, these services can be recombined many times into creative Compositions with least integration effort (Mukhtar, Demystifying Service-Oriented Architecture Part-I: Promised Goals and Benefit, 11/2011).

2.2. Vendor Diversity Options

The word '*options'* is the key to understanding this goal. SOA does not advocate vendor diversity per se. Having multiple vendors is not necessarily a positive thing for an IT enterprise. In fact, unnecessary and excessive vendor diversification can impair an organization's ability and agility (Erl, 2008). However, having the option to substitute vendors when needed is the goal. In other words, SOA seeks to save an IT enterprise from the dreary situation of vendor lock-in. By providing the ability to pick and choose the best-of-breed products and technology innovations, SOA empowers an organization with constant freedom to look for better solutions and incorporating latest technology advancements.

2.3. Federation

Federation implies a unification of disparate environments while still allowing independent evolution. Service-Orientation's ideal vision for the target-state environment includes a Federated Endpoint Layer exposing all services (Erl, 2008, p. 58). This means that no matter how different the underlying logic and implementation of these services might be, at the contract level they are all standardized and harmonized. SOA hopes to accomplish this ambitious task by proliferation of industry and design standards during the design, development and deployment stages of service inventory.

2.4. Business and Technology Alignment

Due to ever-increasing complexity of its products, collaboration among different people participating in the same development project has become the sine qua non of success in the software industry (Arsenyan & Büyüközkan, 2009). Experience shows that the initial business requirements vision is lost when the software architecture is instantiated (Dahman, Charoy, & Godart, Aug 29-Sep 2 2011). In the SOA context, the Business and technology alignment refers to the ability of the information technology architecture to evolve with the changing business. SOA hopes to achieve this alignment through the collaboration of business and IT experts during the analysis and modeling stages. The output of this business and technology partnership manifests in the form of delivered Services with solid business functional-context around them. Business logic is partitioned and shaped into these business services that are designed to be flexible and can continually evolve in tandem with the business.

2.5. Organizational Agility

Business agility could be defined as the flexibility of a business in response to rapid changes in the business environment (Rostampour, Kazemi, Zamiri, Haghighi, & Shams, 2011). SOA promises an IT that is responsive to the changing business. Some experts go so far as to say that "implementing SOA means building for change" (Bloomberg, 2013, p. 47). It hopes to achieve this goal through designing IT assets that are inherently flexible and uniquely designed to be modified with minimal code change. This is accomplished by means of proper decomposition of a given business problem, and designing a solution which is inherently disposed to Service Composition and Agnostic Services (i.e. Utility and Entity type services). Because such services are ideally normalized, – i.e. they don't have overlapping functional-boundaries – necessary changes in the IT assets are minimized (Mauro, Leimeister, & Krcmar, 2010). Any new development requirement can be partially met with the existing IT assets. This ideally positions IT to remain flexible with the growing and moving business.

2.6. Increased Return on Investment (ROI)

An "architecture is defined by its significant design decisions, where 'significant' is measured by the cost of change" (Booch, 2008, p. 18). ROI is the most tangible value in cost-savings that SOA pledges. Even though, in the beginning of an SOA adoption journey, the initial cost of an SOA based solution for an organization is expected to be somewhat higher than a project-specific traditional silo-based solution, the additional initial investment is hoped to be recouped many times over once the Service-Orientation reaches a critical level (Erl, 2008). This is where Service-Orientation has been influenced by the commercial product design. SOA encourages architects to design and create

agnostic solution-logic and shape it into Entity and Utility services that can participate as effective and efficient components of complex and creative Service Compositions.

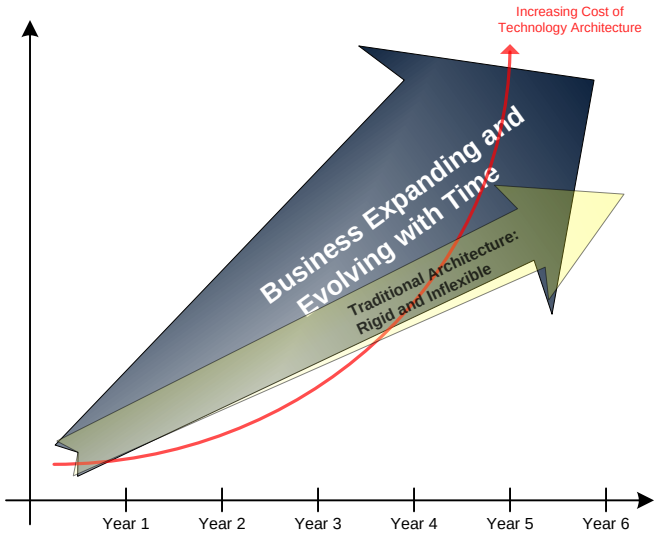


Figure 7: The traditional technology architectures are tactically focused, and over a period of time prove rigid, inflexible and incapable of evolving in tandem with the business. As a result, business and technology architecture grow increasingly out of synch – business requirements fulfillment decreases. At the same time, the cost and effort to bend the existing architecture to comply with changing business increases. (Mukhtar, 11/2011)

2.7. Reduced IT Burden

In the ultimate target-state vision, Service-Orientation envisions IT becoming "less of a burden on the organization and more of an enabling contributor to its strategic goals", a business-partner and an enabler of change (Erl, 2008, p. 64). SOA hopes to reduce waste and redundancy by reducing size and operational costs. After a critical stage has been reached in SOA adoption and maturity, the organization would experience increased responsiveness to changing business needs with reduced overall IT costs. The IT department would become leaner and agile compared to a traditional IT enterprise.

3. Common Measure of Success for an SOA Adoption Initiative

The progress of an organization on the path of SOA adoption, and the attained level of maturity needs to be measured. "Understanding the extent to which SOA solutions conform to the concept of *service orientation* is important. This is particularly true because with such knowledge we might be able to explain the success and failure of SOA implementations" (Aldris, Nugroho, Lago, & Visser, 2013).

Software designers and researchers have been keenly aware of the need to evaluate the level of maturity of an SOA adoption (Cummins, 2009, pp. 19-26). Without an empirical method of measuring progress, there is no objective way of assessing the level of success or failure in any organization with respect to SOA adoption initiatives. In order to address this question of measurability, the vendors and the researcher community have developed several SOA Maturity Models (Gerić & Vrcek, 22-25 Jun 2009).

A typical SOA Maturity Model is a two dimensional matrix where some number (commonly ranging from three to seven) of maturity stages are plotted on the x axis, and some number (commonly ranging from five to seven) of critical SOA adoption factors are plotted on the y axis. This provides a measuring tool for the factors deemed important to be measured individually and independently.

It can be safely assumed that the level of fulfillment of business benefits from an SOA adoption would naturally correspond with the level of SOA maturity achieved at an organization. In other words, an organization with relatively low SOA adoption maturity experiences the realization of relatively fewer benefits, as opposed to an organization with relatively high SOA adoption maturity.

4. Success-Rate with SOA

SOA seems to be failing (Appendix B), and its adoption has produced only mixed results (Welke, Hirschheim, & Schwarz, 2011). Compared to some spectacular results in terms of SOA adoption that produced measurable business results, some disappointing failures are also evident (Gottschalk & Solli-Sæther, 2009; Sulong, 2013). Even after struggling for many years and spending a fortune, some organizations just couldn't get SOA to work (Appendix G). As a result, "the glorious SOA utopia" in terms of strategic business benefits that are promised by this unique design paradigm have not fully materialized for some organizations, even after substantial investment in terms of time and resources (Faust, 2010). An objective analysis of the situation is worth pursuing.

In 2009, a much debated article appeared in a blog that claimed "SOA is dead" (Appendix E). The ensuing discussion, and the existing mixed business results, accumulated much negative goodwill around SOA design paradigm in the industry. However, many industry experts rushed to its defense (Bloomberg, 2013), and explained "what went wrong" (Appendix D).

Researchers and practitioners cannot pinpoint the reason(s) with certainty. Some point to its inherent complexity (Kral & Zemlicka, 2009; Mamaghani, Mousavi, Hakamizadeh, & Sadeghi, 23-25 June 2010) while others point to the widespread misunderstanding and confusion this over-used buzzword has acquired in the industry (Bloomberg, 2013; Gerić & Vrcek, 22-25 Jun 2009). One thing most experts agree is that SOA adoption is non-trivial (Kannan, Bhamidipaty, & Narendra, 2011). A consensus seems to be converging on the idea that SOA might be largely misunderstood. People who claimed to be doing SOA were in fact not really doing SOA but were really getting

confused by the vendor hype (Appendix C). If this is true, then an accurate measure of SOA adoption maturity becomes increasingly important in order to separate the actual failures from the mere perception of failure.

4.1. Challenge of Measurability of SOA as an Architectural Style

Despite the profusion of SOA maturity models, empirical studies show that "few [organizations] could measure whether they are" having any real success (Welke, Hirschheim, & Schwarz, 2011, p. 63). Thus the ability to objectively measure the SOA adoption maturity seems to be eluding us.

An effective system development consistent with principles of SOA is non-trivial (Selmeci & Rozinajova, 2012). Most of its complexity comes from the grassroots level – the architectural decisions that must be made while shaping individual services in light of the *Principles of SOA*. The application of these SOA principles, however, is a subtle matter, and the assessment of SOA maturity is not necessarily obvious (Kannan, Bhamidipaty, & Narendra, 2011). In reality, an organization could be working under the illusion of being SOA compliant, while in fact, only perpetuating the traditional silo-based architecture under a veneer of web-services and some expensive Commercial of the Shelve (COTS) products like an Enterprise Service Bus (ESB) and a Service Registry and Repository (SRR) (Bloomberg, 2013, p. 23). If that were the case, the grand promises of SOA like increased ROI, increased organizational agility and reduced IT burden, cannot materialize (Erl, SOA: Principles of Service Design, 2008, p. 56). The ambitions of such an organization towards SOA adoption will only end in confusion and frustration. Some of the recent surveys show that several organizations are pulling back from SOA after struggling with SOA adoption for a few years with minimal results (Appendix G). If the

problem lies in the understanding of the SOA design paradigm at an organization then the existing SOA maturity models can only produce skewed results in terms of measuring SOA adoption maturity.

The real and meaningful SOA adoption happens at the individual service design level (low-level), and that is what is needed to be measured. To use the analogy of bricks and wall, if a complete SOA eco-system is a wall, then each service is a brick in that wall (Dubey, 2010). The aggregate strength of the wall depends on the strength of the individual bricks. The existing SOA maturity models measure the strength of the wall, but ignore the bricks. The challenge, thus, is to come up with a Service Architecture Maturity Model (SAMM) for SOA adoption that can accurately and empirically measure the strength of each brick, and as a result, provide a fair assessment of the strength of the complete wall. Such would be an accurate measure of SOA maturity in an organization.

The general hypothesis under the given circumstances is that the perception of SOA failure at some organizations could simply be due to the use of high-level maturity models for measuring the SOA maturity. For the sake of understanding, consider the following hypothetical but illustrative scenario:

A large public sector organization has spent 20 million dollars over a period of last six years on an SOA adoption initiative. At the end of this period, the CIO polls the heads of all business units that the IT department serves, in order to measure business results from his investment in the SOA program. On the scale of 1 to 5 (5 being the highest), the average score comes to a disappointing 0.75. The CIO asks the technical advisor to provide the current maturity level of SOA adoption at the agency, using one of the

existing maturity models, in order to rationalize the low score received from the business units. Surprisingly, the maturity level measurement, on the same scale, comes to an impressive 4.0. The critical question then becomes: if SOA is well mature at this agency, why such modest business-results? Considering the two contradictory pieces of the information, the CIO concludes that SOA is an empty-promise, or at least not workable at the agency. Disillusioned, the CIO pulls back all the future investments planned for the SOA adoption program, and writes-off $20 million already spent.

The maturity model used for assessing the SOA maturity in the above scenario was a high-level model, focused on processes, tools, technologies and infrastructure, but not on the low-level service architecture, which Manes refers to as "Micro SOA" (Manes, 2013). A reasonable hypothesis could be that the maturity results are misleading and confusing because of the wrong scope of measure. Next, the CIO of the same organization is asked to use the proposed *Service Architecture Maturity Model* (SAMM) to measure the low-level or micro SOA maturity at the same organization. This time, the maturity model produces an SOA maturity score of 1.0, well aligned with the minimal realization of business results. From this analysis, the CIO understands that there's something wrong in their understanding of SOA, and sets out to fix the problem instead of blaming the SOA design paradigm for the realization of inadequate results.

4.2. Specific Research Questions

SOA is a promising design paradigm but some organizations continue to struggle on the SOA adoption path. There could be a variety of reasons for that; however, it seems logical that the achieved maturity in SOA adoption should be positively correlated with the derived benefits from SOA adoption in an organization. In other words, the more

advanced an organization is in SOA adoption, the more noticeable should the benefits be from such an adoption, at least to a certain degree. If, however, a discrepancy exists between these two factors, can it be reconciled?

Based on the above discussion, for such an organization, the following set of research questions arises.

a. Is SOA largely misunderstood at an organization that struggles in reaching a reasonable level of SOA adoption maturity, and in producing comparable business results?

b. Is the lack of measure of SOA adoption maturity at the Service Architecture level a major cause of the perceived failure of SOA design paradigm?

c. How to measure the level of SOA adoption maturity at the Service Architecture level?

d. How can this more pointed maturity-measure of SOA adoption actually help an organization progress to a higher maturity stage?

Chapter 2: Literature Review

The questions raised in the previous section fall into two related but separate areas:

1. The existing SOA maturity models that are commonly used in the industry to measure the SOA adoption maturity
2. Service architecture based on the Principles of Service Design

The existing literature of both these areas is reviewed under separate sections below.

1. A Critical Review and Analysis of the Existing SOA Maturity Models

Software researchers have been keenly aware of the need to evaluate the level of maturity of SOA adoption. Without an empirical method of measuring progress, there's no objective way of assessing the level of success or failure in any organization with respect to SOA adoption. In order to address this question of measurability of SOA adoption maturity, the vendors and the researcher community has come up with several SOA Maturity Models. Some of the important ones are discussed below.

1.1. SOA Maturity Model by SOA Alliance

In 2006, the Object Management Group (OMG) published an SOA Reference Model by SOA Alliance – a group of independent SOA Practitioners – which included a high-level SOA maturity model (Appendix H).

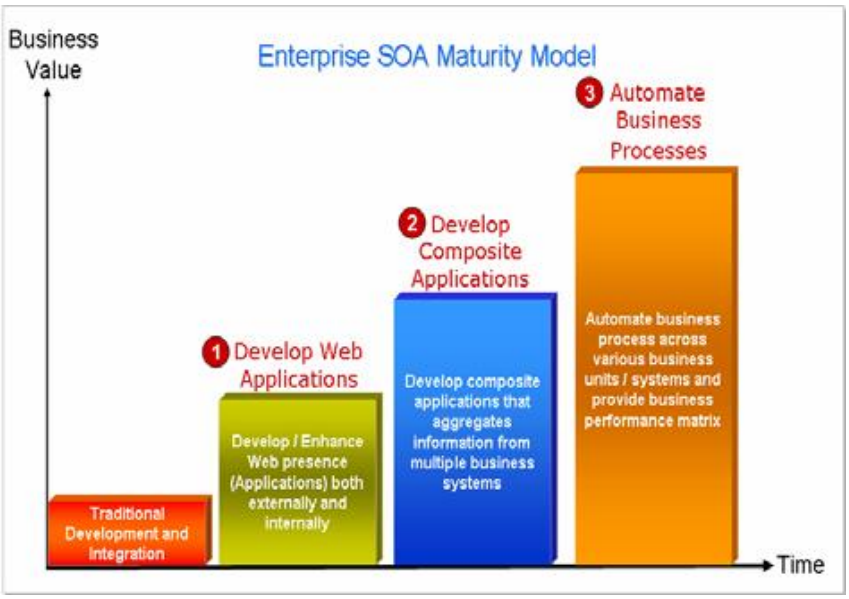


Figure 8 – Enterprise SOA Maturity Model by SOA Alliance (Appendix H)

As can be seen from Figure 8, this model sliced SOA maturity into the following three stages.

1. **Web Application Development**: At this early stage, organizations provide browser-based business solutions to both internal and external users, usually in the form of web-based CRM, ERP or custom applications. Enterprise services such as content management, search and instant messaging are also usually built during this stage.
2. **Develop Composite Applications**: At this middle stage of SOA maturity, organizations improve their data quality. Building on that improvement, IT applications provide aggregated information from multiple sources to first internal users, and later to external users as well.
3. **Automate Business Process**: At this highest level of maturity, the applications, data and infrastructure all work in harmony and empower the user with the right information at the right time. This level of SOA maturity results in increased ROI through the consolidation of multiple business systems. Organizations transform to

the target-state of end-to-end business process management, rather than traditional point-to-point solutions.

This maturity model describes the three stages from which organizations go through as they make progress on the SOA adoption track. The model does not attempt to describe an assessment process of the internal architecture of the services, or a method of measurement of the level of service-orientation achieved by an organization.

1.2. SOA Maturity Model by IBM

IBM architects find it meaningful to describe the SOA maturity in seven distinct stages in their Service Integration Maturity Model (SIMM) (Arsanjani & Holley, 2006). This model considers de-coupling and agility as determining factors, delineating one stage from the other. The seven stages are listed below.

1. Silo (data integration)
2. Integrated (application integration)
3. Componentized (functional integration)
4. Simple services (process integration)
5. Composite services (supply-chain integration)
6. Virtualized services (virtual infrastructure)
7. Dynamically reconfigurable services (eco-system integration)

At level one of this SOA maturity track, the systems integration is ad-hoc and inflexible because of the use of proprietary technologies. Moving up to the level two means embracing some systemic adoption of EAI in order to leverage parts of legacy applications and data integration. The level three is about modularization of applications

which solidifies the interfaces and contracts between components, fostering decoupling. The level four on the SOA maturity path is when organizations first start acting as provider of services to its internal and/or external consumers, on a small scale. The level five is signaled in when the service eco-system of an organization is mature enough that it can support on-demand interaction with its trading partners. Level six focuses on the virtualization of infrastructure to be used by the service implementations in order to make it agile. And finally, at level seven, organizations reach an architectural maturity that they can now dynamically reconfigure service compositions at runtime using externalized policies and monitoring.

The natural progression of an organization through the above listed levels of SOA maturity is incremental which is depicted in the Scope of Adoption diagram shown in Figure 9. SOA adoption commonly evolves up from the project level into general adoption of technologies at the Line of Business (LOB) level. This involves some assessments and working proof-of-concepts. As the technology adoption gets mature, and business sees value in sharing service capabilities across business domains, the benefits in terms of reduced cost and complexity start becoming visible. As the next incremental step, the adoption of shared services gets business buy-in at the enterprise level, and an organization gets consolidation of business functions across several business domains. Finally, the organization is transformed completely – standards get established at the enterprise level, and governance becomes a function of the enterprise. A shared funding model gets established for shared services implementation initiatives.

Figure 9 – Service Integration Maturity Model by IBM (Arsanjani & Holley, 2006)

As can be seen in Figure 9, the IBM's SOA maturity model describes the natural progression of organizations through SOA adoption at relatively fine-grained levels. There are distinct signs that signal the level of adoption of an organization. However, it does not stress nor elaborate the maturity of individual service's architecture, much less point specific criteria to evaluate architecture at that low-level service architecture.

1.3. SOA Maturity Model by Sonic Software

A relatively straightforward SOA maturity model was published by a vendor consortium in 2006 for benchmarking the SOA implementation and planning (Appendix A). This model provided five progressive levels of maturity, from 1 to 5, where level 5

was considered most mature, as can be seen in Figure 10 (Sonic Software Corporation, AmberPoint Inc., BearingPoint, Inc., Systinet Corporation, 2005).

At Level 1, called *Initial Services*, "an organization creates definitions for services and integrates SOA into methodologies for project development". Here a project typically employs an Enterprise Service Bus (ESB), and creates simple SOAP and HTTP Web service.

At Level 2, called *Architected Services*, organizations typically set standards for SOA governance, and SOA infrastructure like an ESB and a Service Repository are established to foster reuse of services.

At level 3, called *Business Services and Collaborative Services*, organizations optimize their business processes through the introduction of Business Process Management (BPM). This is geared towards increasing organizational agility in response to business change.

Maturity Level 4, called *Measured Business Services*, focuses on the collection and dissemination of appropriate performance data to the business users in a continuous feedback loop.

And finally, at Level 5, called *Optimized Business Services*, "business-optimization rules are added, and the SOA becomes the nervous system for the enterprise". At this highest level of SOA maturity, business processes become dynamic, intelligent and event-driven.

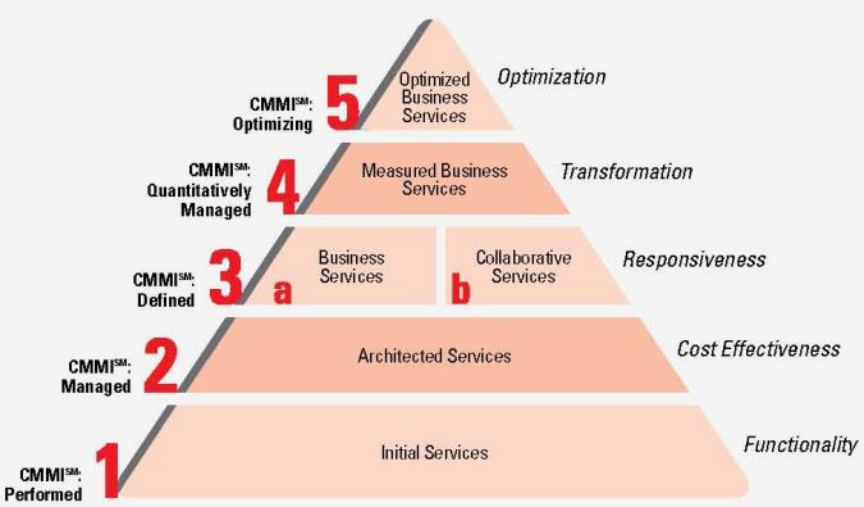


Figure 10 – SOA Maturity Model by Sonic Software (Sonic Software Corporation, AmberPoint Inc., BearingPoint, Inc., Systinet Corporation, 2005)

This SOA Maturity Model claims to provide "a framework for IT and business users to properly evaluate the applicability and benefits of SOA in an organization". It does mention Architected Services at its level two, but does not provide a method to evaluate the maturity of the micro SOA, i.e. at that low-level service architecture.

1.4. SOA Maturity Model by Microsoft

A more elaborate SOA Maturity Model was published by Microsoft Services (Figure 11) under the title of "*Assessment and Roadmap for Service Oriented Architecture*" in December of 2008 (Microsoft Services, 2007). This paper was aimed at providing "a decisive [and] vendor independent perspective on ... SOA capabilities [through] a tailored SOA roadmap with prioritized recommendations, supported by documentation of comprehensive enterprise SOA assessment findings, the SOA Maturity Model workshop, and a services inventory with dependencies and adoption levels".

The SOA Maturity Model that it proposed accounted for 36 technology-independent capabilities as guidance "for what is possible and what is required [in order] to realize the value of a service-oriented approach". This model is shown in Figure 11, and summarized below:

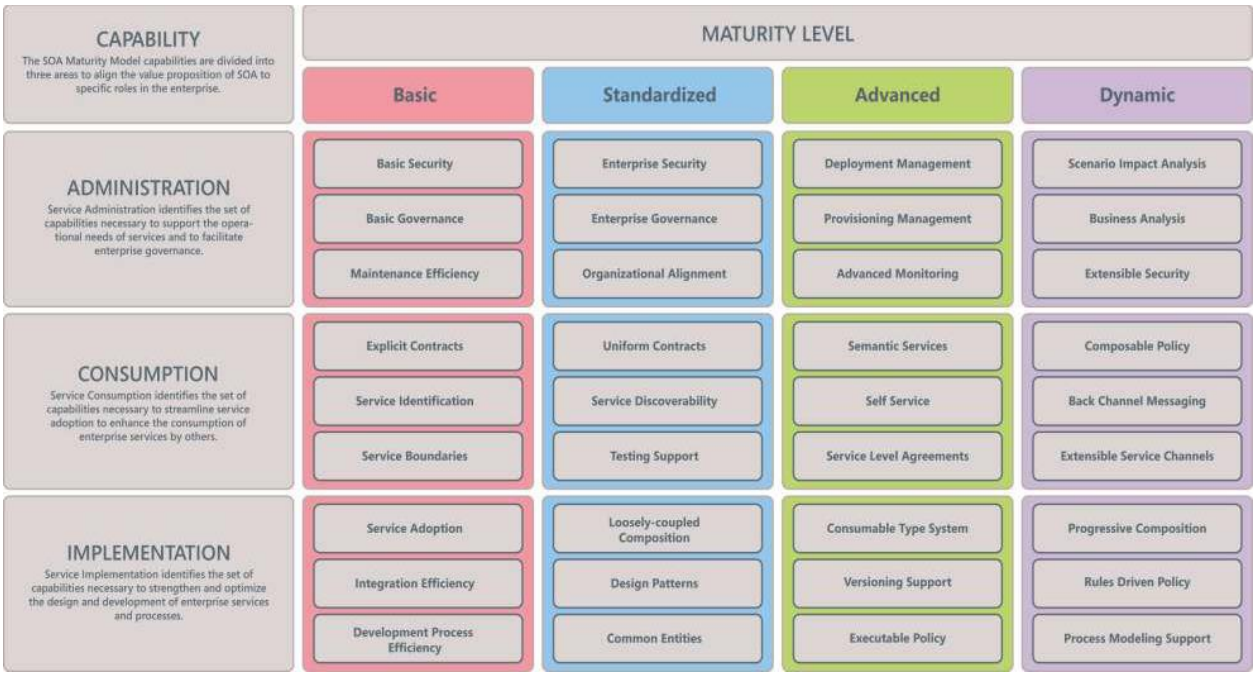


Figure 11 – SOA Maturity Model by Microsoft (Microsoft Services, 2007)

As can be seen from this two dimensional Microsoft's roadmap to "SOA excellence", there are four Maturity Levels measured on the horizontal axis, and three Capabilities on the vertical axis. This implies that, for instance, an organization could be at Basic maturity level with regards to Administration capabilities, but at the same time, on a higher maturity level, say Advanced, for Consumption and/or Implementation capabilities. This independence in capability measurement maturity levels provides some flexibility in the model.

The scope of the maturity measurement in this model, however, remains very high-level. For instance, at the Standardized maturity level, the model provides three measurements under Implementation capabilities: 1. Loosely Coupled Compositions, 2. Design Patterns and 3. Common Entities. It does not say, for instance, anything about how to measure the application of the Design Patterns at the low-level of the service architecture.

1.5. SOA Maturity Model by The Open Group

The Open Group is a large consortium of diverse IT professionals including vendors, customers, consultants, independent architects, academics and researchers. The Group's vision is "Boundaryless Information Flow™ achieved through global interoperability in a secure, reliable and timely manner". An important part of its work involves "achievement of business objectives through IT standards".

The Service Integration Maturity Model (OSIMM) by this Group is a vendor neutral, comprehensive and extendable SOA maturity model which can be used to assess the degree of service integration maturity (The Open Group, 2011). Compared to the other vendor-based SOA maturity models that are mostly tools, technology and infrastructure focused, OSIMM is relatively more inclined towards measuring service-orientation. The Open Group describes its model as below:

OSSIMM is "a model that enables estimation of the degree to which an organization or enterprise has taken up the principles of SOA within their IT and business. There are seven levels, Level 1 being the least take-up and Level 7 being the greatest take-up. Higher degrees of maturity are likely to lead to a higher degree of agility in the business,

but are not necessarily “better”, as each organization may have an ideal level of maturity depending upon their business requirements and business and IT context.”

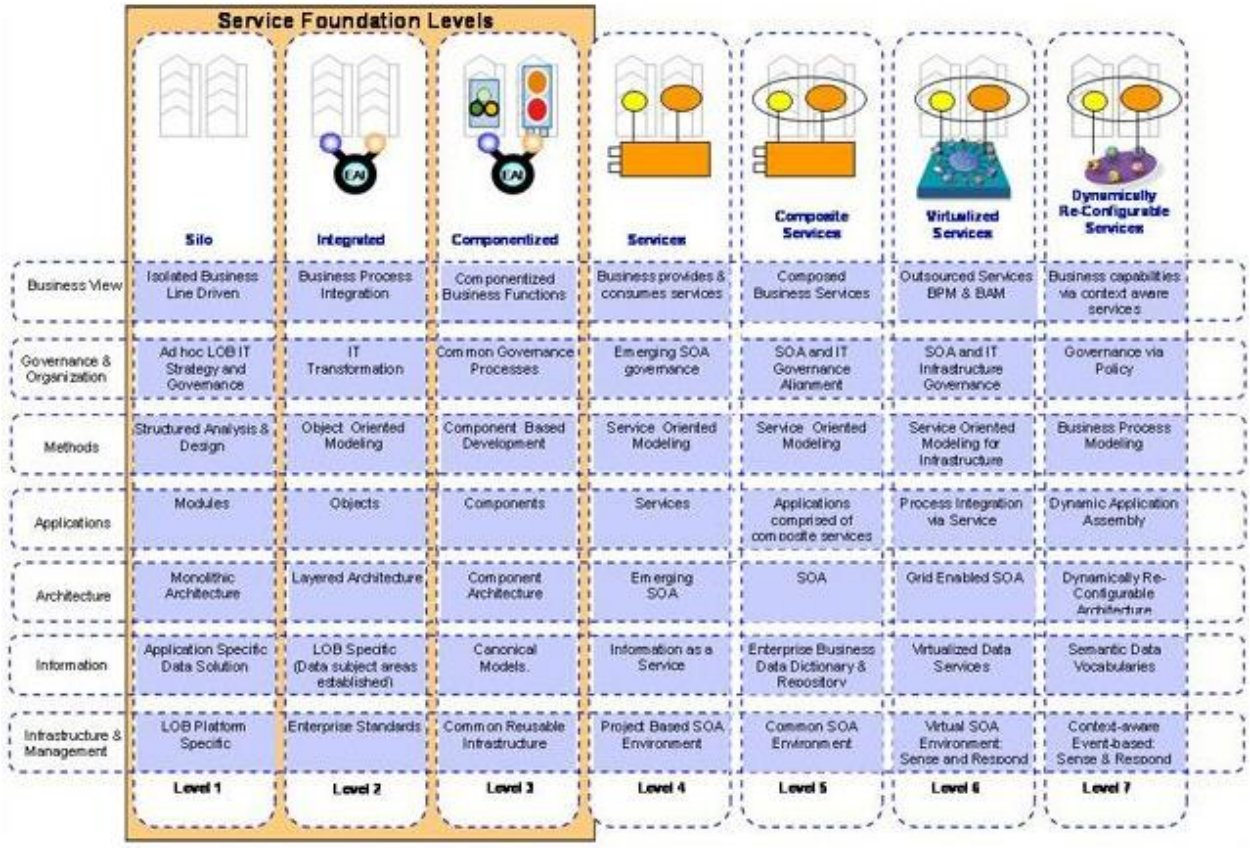


Figure 12 – SOA Maturity Model by The Open Group (The Open Group, 2011)

In the above seven by seven matrix (Figure 12), the columns correspond to the Maturity Levels, and the rows correspond to the Dimensions. Each cell thus defines the maturity level for each of the Dimensions of the SOA adoption. The overall SOA maturity of an organization is assessed by aggregating its Maturity Level in each Dimension. Since the Dimensions and the Maturity Levels of this model are well described at the source (The Open Group, 2011), those are not reproduced here. Instead, the distinguishing features of this model from the other vendor-based maturity models are

discussed below. From the perspective of the current analysis, the four most important features of this model are:

a. Focus on service-orientation – Instead of technology infrastructure
b. Low-level consideration of service architecture, i.e. micro SOA
c. Assessment method and questionnaire
d. Extensibility

Focus on service-orientation: SOA is a unique architectural style that is focused on service-orientation, i.e. a way of thinking in terms of services. Relative to the other SOA maturity models, OSIMM provides some focus on the evaluation of an organization's maturity level based on the principles of service design. For instance, the assessment questions under the Architecture dimension asks: "What architectural principles define your approach?" Under the Application dimension asks: "How common is re-use in your organization?" Granted that it's not going far enough to the level of the specific Eight Principles of Service Design, but it is a step in the right direction. The Dimensions of this model still include, for instance, Governance and Organization and Infrastructure and Management which shifts the focus away from the micro SOA and the low-level services architecture.

Low-level consideration of service architecture: Relative to the other SOA maturity models, OSIMM shifts some focus in its SOA maturity assessment back to the basics of micro SOA. This point becomes noticeable by reviewing the seven maturity levels of this model. Most of the seven levels (e.g. level 3, 4, 5 and 6) try to focus on the issues related

to the service decomposition which makes the overall architecture naturally amenable to the application of SOA principles at the low-level service architecture.

Assessment method and questionnaire: In addition to the maturity matrix, under the *Assessment Method,* chapter 10, OSIMM also provides a detailed evaluation methodology (The Open Group, 2011). Under each of the seven dimensions, it provides a set of assessment questions, maturity indicator, and a mapping to the maturity attributes. For instance, Architecture dimension provides Base Model Maturity Indicator as: "…by identifying those service components that have been designed and are deployed using formal SOA methods, principles, patterns, frameworks, or techniques."

Extensibility: The base OSIMM model can be extended by adding additional maturity indicators, assessment questions, and corresponding attribute mappings. While the principles of service-orientation are relatively more stable than the implementation technologies, there's room for improvement and change in both areas. To this end, the OSIMM is intentionally kept open to new and evolving techniques for implementing services such as cloud computing. The extensibility of the OSIMM framework is intended to provide a method to augment the base OSIMM model to include such concepts.

Due to the above mentioned characteristics, compared to the other existing models, OSIMM comes closest to an SOA maturity assessment model that can be extended and modified for building a low-level Service Architecture Maturity Model.

1.6. SOA Maturity Evaluation by Eric Marks

A veteran author on SOA, Marks wrote a small blog in August 2006 titled "A Test of Maturity" (Marks, 2006). Without providing a full-blown SOA maturity model, he

mentioned a few key items in his maturity evaluation criteria: SOA strategy and vision; services concept and maturity; SOA architecture and technology stack; SOA governance and policy enforcement; organization and culture; and SOA metrics and results. What is important from the perspective of the current analysis in this "test of maturity" is the fact that under Services Concept and Maturity, Marks stresses on the centrality of the concept of Service in SOA design. He states that an organization's understanding of "service…is the vehicle through which [it] attains the business goals articulated in [its] SOA strategy". However, he does not elaborate on this idea further as to how service architecture can or should be used to measure an organization's SOA maturity.

1.7. SOA Maturity Model by Business Process Trends (BPTrends)

Business Process Trends is a monthly webzine (web based magazine) that publishes articles and reports on Business Process topics. It provides guidance and direction on new developments and trends in the field of business process change. In 2007, it proposed its own SOA maturity model which closely followed on the lines of CMMI levels as shown in the Figure 13 below (Appendix I).

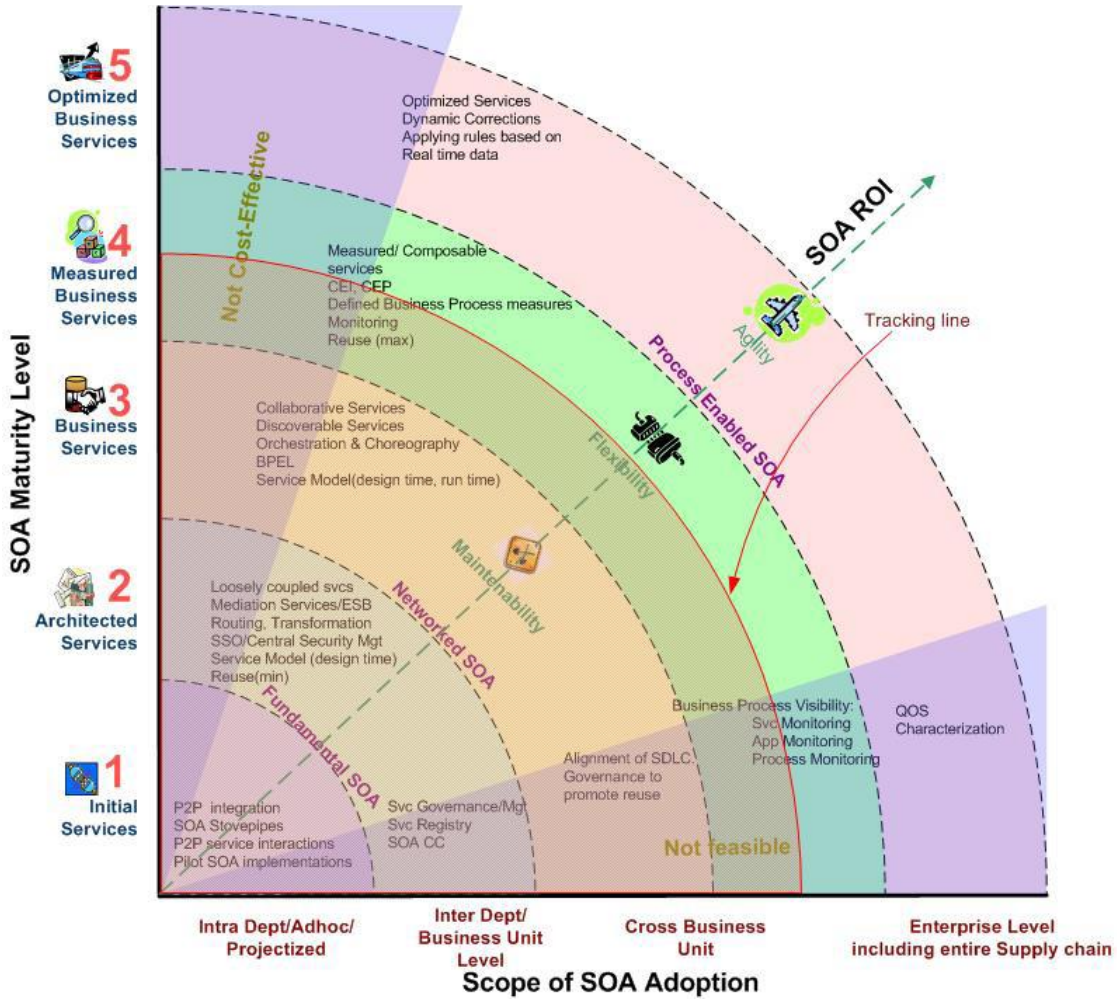


Figure 13 – SOA Maturity Model by BPTrends (Appendix I)

The authors of this model stress that in order to evaluate SOA maturity of an organization, it is important to have a multi-point view that encompasses as many aspects of the organization's SOA implementation as possible, to arrive at its true state of SOA maturity.

The measure of the Scope of Adoption along the x-axis, and the five levels of SOA maturity on the Y-Axis in this model are self-explanatory. The details of each scope and level are available at the source (Appendix I). The important distinguishing features of this model are briefly discussed below.

On the question of ROI (Return on Investment) from SOA adoption, the model depicts a gradual progress as the SOA maturity increases as shown in the quadrant section. Increased maintainability is followed by increased flexibility and ultimately, agility at the enterprise level is achieved at the highest level.

On the important point of cost effectiveness and feasibility, this model presents the shaded areas to represent non-cost-effective and infeasible areas. This feature uniquely captures an important fact about SOA adoption, i.e. these areas do "result when the level of service maturity does not keep up with the degree of SOA adoption. For example, implementing process enabled SOA for intra-department needs may not be cost-effective. Similarly, trying to employ fundamental SOA techniques to achieve the goals of enterprise level SOA is not feasible." (Appendix I)

1.8. SOA Maturity Model by Welke, Hirschheim and Schwarz

In 2011, also following the lead from the Software Engineering Institute's (SEI) Capability Maturity Model Integration (CMMI), Welke et al. offered an SOA maturity model with five similar Maturity Levels, which they propose to be termed as "capability orientation model". Against these five Maturity Levels, laid out vertically, this SOA maturity model mapped six Dimensions horizontally. "These levels use the same basic CMMI terminology but reflect the changing locus of motivation for SOA adoption. Each maturity level indicates the principal capability needed to achieve a higher-level capability – that is, one that moves away from IT-dominated reasons for SOA use toward enterprise-level transformational objectives" (Welke, Hirschheim, & Schwarz, 2011). In order to distinguish the standard CMMI maturity categories from their model's

"Orientation View", the authors include "SOA view" as the primary attribute distinguishing the maturity progression.

In the author's estimation, these six dimensions (SOA view, Benefits and metrics, Business involvement, Methodology, Service sourcing, and Governance) represent the key success factors in measuring SOA adoption maturity of an organization.

Table 1. SOA maturity model.

Maturity level	SOA view	Benefits and metrics	Business involvement	Methodology	Service sourcing	Governance
Initial	Fine-grained software components	Promise of reuse; no metrics	Hidden from business	Minor adaptation of current software methods	Largely derived from existing application programming interfaces	Basic service definition policies at project level
Managed	Emerged software architecture	Standardization of data and resources	Services exposed as part of project cost	Project-level service definition methods	In-house development of new fine-grained software components	Service policies managed by registry monitors
Defined	Business process support	Business process redesign	Services part of requirement capture	Business service definition methods	External sourcing possible by defined function	Full set of service policies and metrics in place
Quantitatively managed	Enterprise service architecture	Agility and flexibility	Organizational "service thinking"	Intra- and inter-organizational service definition	Enterprise service bus to coordinate services	Business and IT governance metrics aligned
Optimized	Adaptive architecture	Autonomic systems	Value-stream "service thinking"	Value-chain service optimization	Full integration up and down the stack	Policy feedback used to adjust delivery

Figure 14 – SOA Maturity Model by Welke, Hirschheim and Schwarz (Welke, Hirschheim, & Schwarz, 2011)

As shown in the above five by six matrix (Figure 14), against each of the five Maturity Levels (vertical axis), the maturity of SOA adoption increases as it moves from top to bottom on any given SOA Dimensions (horizontal axis) independently, with each Level indicating the "principal capability" to reach to the next higher level. The authors suggest that a higher maturity level indicate an organization's progression to a more mature SOA eco-system, and correspondingly, results in a higher realization of anticipated business benefits. This model was offered as an evaluation tool for the organizations interested in measuring their achieved level of SOA maturity, and for

suggesting helpful steps to further progress on the SOA adoption path. Finally, the authors do "encourage other researchers to further develop [their] model with both qualitative and quantitative measures." (Welke, Hirschheim, & Schwarz, 2011).

Not unlike other existing SOA maturity models, this model too fails to treat SOA as an architectural style, and does not attempt to measure its adoption maturity based on the service design principles. Instead, this model attempts to measure the SOA adoption maturity across the six dimensions that were "considered" significant and essential by the authors.

1.9. SOA Maturity Model by Rathfelder and Groenda

Explaining how the "existing SOA maturity models provide only weak assistance with the selection of an adequate maturity level", and that "most of them are developed by vendors of SOA products and often used to promote their products", in 2008, Rathfelder and Groenda offered a new SOA maturity model, which they named, "Independent SOA Maturity Model" or "iSOAMM". By "independent" the authors meant that their model is independent of the "technologies and products" that often become inseparable part of SOA infrastructure and platforms. The authors also claimed that this model will "enables enterprises to select the most adequate maturity level for them, which is not necessarily the highest one". The overview of this model is reproduced in Figure 15 below.

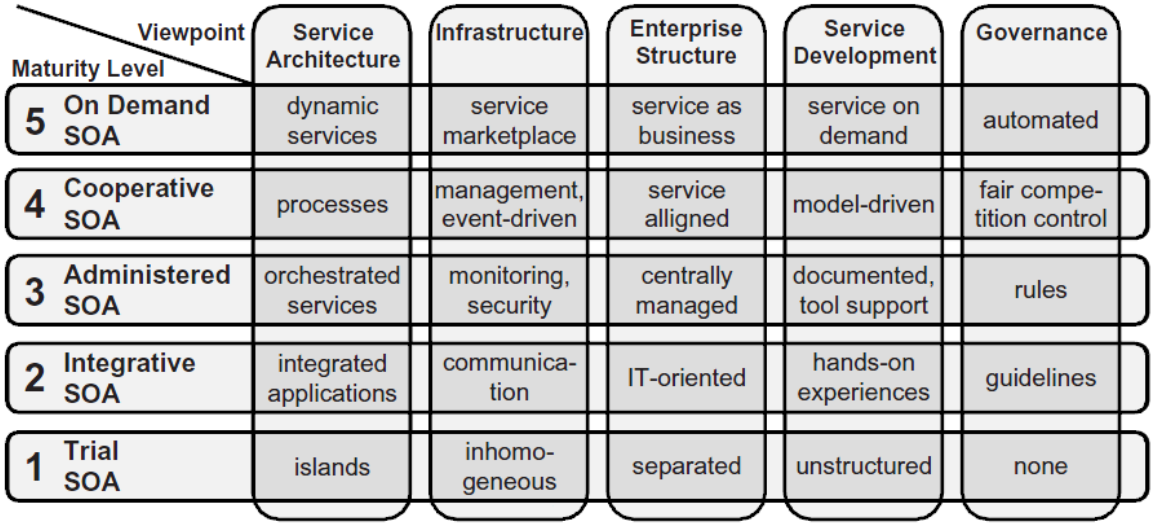


Figure 15 – SOA Maturity Model by Rathfelder and Groenda (Rathfelder & Groenda, 2008)

As can be seen in the above five-by-five matrix, this model defines five maturity levels (numbered 1 through 5, 5 being the highest, in five rows) looked from five different viewpoints (columns): service architecture, infrastructure, enterprise structure, service development, and governance. The authors of this model considered these five viewpoints critical enough to be included in the model, to be used as yardstick for the SOA adoption maturity progress.

From the perspective of evaluating SOA as purely an architectural style, in this model, except for the first viewpoint, i.e. service architecture, the other four only add to the confusion, and lose the focus on measuring the maturity of the architecture. Moreover, even within the relevant "service architecture" viewpoint, the model fails to go deep enough at the principles of service design level in order to measure the true maturity of the SOA design paradigm.

1.10. Challenge Not Fully Addressed by the Existing Maturity Models

While it can be agreed that a comprehensive view might be helpful in assessing overall SOA adoptions, for many organizations, more focused attention is needed to the underlying service-orientation design principles to measure the SOA maturity. As Aldris et al rightly argue that "in the end, the design of an SOA implementation will determine the sustainability of the implemented solution in supporting the business goal" (Aldris, Nugroho, Lago, & Visser, 2013).

A review of the above discussed maturity models brings out a critical point: the existing SOA maturity models and matrices mostly remain focused on SOA enabling tools, technologies, infrastructure, management and processes, while ignoring the internal and essential attributes of SOA, i.e. the service design principles. These kinds of high-level maturity models might measure the maturity of an SOA technology platform, but not the maturity of "SOA as an architectural style" (Aldris, Nugroho, Lago, & Visser, 2013). To a certain degree, this reflects part of the prevalent misunderstanding about SOA design paradigm. The existing SOA maturity measurement models miss the fact that "SOA is both technology and protocol neutral" (Bloomberg, 2013, p. 21) and that the real SOA happens at the service-architecture level (Mukhtar, 12/2011). Measurement of SOA adoption maturity on the bases of the *eight Principles of SOA*, all of which focus on shaping the individual services, is the critical gap in the existing research. In the absence of such a yardstick, the claims of progress (or failure) on the road of SOA adoption will remain tenuous. This existing gap justifies creating a new low-level micro SOA maturity model, capable of measuring the maturity at the service architecture level, and can be called the Service Architecture Maturity Model (SAMM).

2. ***The Service Architecture Under the SOA Principles of Service Design as Conventional Software Design Best-Practices***

Designing software is a creative activity, and thus, like other creative activities, there is no absolute formula which guarantees a good design. However, there are some known principles that could increase the probability of attaining an effective design (Bahill & Botta, 2008). A software design principle may be defined as 'an adopted rule or method for application in action'. Design principles should help generate ideas and enable an architect to think through design implications. Most software design principles and practices tend to be rules of thumb rather than hard-and fast rules (Wirfs-Brock, 2009).

Thinking in terms of software design is as old as software itself. As Ward explained, it is "simply logical ways of going about designing a system. The fundamentals include modularity, anticipation of change, generality and an incremental approach. Modularity refers to the division of the system into smaller, more manageable components. The aim is for each component to have high cohesion (e.g., just do one thing) and low coherence (e.g., that it is not highly intertwined with other components). Anticipation of change means that the system should be adaptable as no system is static and there is always the possibility of change. Generality means that designers should investigate whether there is a more general solution to the current problem and that, by providing a more general solution, the designers will be providing solutions to other usage situations. *Incrementality* refers to the fact that often the system does not have to be delivered as one large piece of software" (Ward, 2006).

The science of software architecture is primarily concerned with the description of programs. It works on different hierarchical levels, and can be seen as a continuum of software architecture, design and implementation (Eden, Hirshfeld, & Kazman, 2006). In the field of software engineering, architectural decisions are often considered "hard to make" (Fowler, 2003) and costly to change (Booch, The Economics of Architecture-First, 2007). It is well understood that the approaches for modeling and implementing service-oriented systems are founded on the same fundamental ingredients of computing systems as other paradigms – namely, data and operations (Atkinson & Bostan, 2009). Although, the design principles may vary significantly depending on the application for which the software is intended (Simonelis, 2004). In order to successfully achieve the promised benefits from the SOA adoption, organizations should understand what SOA really is and which key attributes are crucial in implementing SOA solutions (Aldris, Nugroho, Lago, & Visser, 2013; Stal, Mar-Apr 2006).

In SOA design paradigm, a Service can be defined as a unit of solution logic to which service-orientation has been applied to a meaningful extent (Erl, 2008, p. 28). This Service is a physically isolated piece of software with some special design characteristics, a distinct *Functional Context* and a set of related *Capabilities*. The key architectural decisions that go into shaping an individual service are commonly known as the principle of service design.

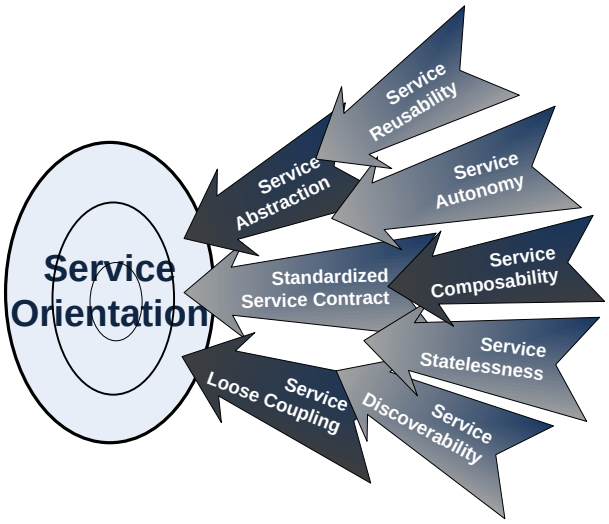


Figure 16: The Eight Principles of Service Design – five that directly result in the desirable characteristics of SOA, and three that act as regulators of the other principles. Applied together in a judicious and balanced way, these principles make a solution truly Service-Oriented (Mukhtar, 12/2011).

Although some researchers still account for less (Ahmed & Ahmed, 2013), after a phase of disagreement (Legner & Vogel, 9-13 July 2007), the industry consensus seems to be towards eight accepted principles of service design as illustrated in Figure 16. Application of these principles "collectively define Service-Orientation as a design paradigm", and should be looked at as a spectrum (Erl, 2008). There exists an intricate internal dynamic among these principles, in which these principles are interdependent and intertwined. Only a holistic view and a balanced application of all eight principles produce best results towards Service-Orientation. Unless the principles discussed below are applied to a reasonable extent, the promised strategic benefits of SOA adoption will not materialize, irrespective of the technology used or vendor employed (Erl, 2008, p. 107).

Long before SOA was accepted and established as an industry best-practice design paradigm, software engineers were acutely aware of the design principles that now

represent the essence of SOA. In other words, SOA did not emerge out of the blue; it rather evolved and coalesced under the influences of existing software design paradigms. Before reviewing each SOA principle, the historical underpinnings in the conventional software design models are discuss below.

2.1. Standardized Service Contract

2.1.1. Historical Underpinning in Conventional Software Design

Deliberating the fundamental principles of good system design, Bahill & Botta advise that “special care should be given to interface design so that the interface does not have to change when its associated systems change” (Bahill & Botta, 2008). Going further back, quoting from their predecessors, they also recommend that “different entities should use the same interface, rather than having a specialized interface for each entity” (Schultz, Fricke, & Igenbergs, July 2000). Software engineers have been aware of the significance of the idea of “Design by Contract” (DbC) since at least late 1980s. This well-understood principle advocates defining formal, precise and verifiable interface specifications for software components. These specifications are referred to as "contracts", in accordance with a conceptual metaphor with the conditions and obligations of business contracts (Meye, Oct 1992; Meyer, 2007). With this historical backdrop, it’s possible to see how SOA adopts and extends the principle of Standardized Service Contract from its predecessor design paradigms.

2.1.2. Definition and Discussion

The principle of Standardized Service Contract may be defined as "Services within the same service inventory are in compliance with the same contract design standards" (Erl, 2008, p. 130)

Services should express their capabilities via a standardized service contract, thus *Contract-First* design process is strongly advocated by this principle. The purpose is to ensure that the manner in which services express their functionality and represent their internal data-types continue to confirm to an enterprise (or domain) inventory standards. Increased consistency of expression, of both data and functionality, is the specific characteristic sought by the application of this principle, and is geared towards achieving the Federated End-Point layer goal (Mukhtar, 12/2011). Some experts go so far as to say that "the essence of the SOA style is the decoupling of service consumer and service provider via the service contract" (Zimmermann, 2011). It requires specific considerations while designing a service contract in terms of both the quality and the quantity of the published content (Curbera, 2007). The emphasis is on the expression of functionality, definition of data types and data models, and assertion of policies by the services in a standardized fashion (Figure 17). Through this design principle, a constant focus remains on ensuring that service contracts are optimized, standardized and are appropriately *granular* so as to ensure that the endpoints are consistent, reliable, and governable in pursuance of the Increased Federation goal.

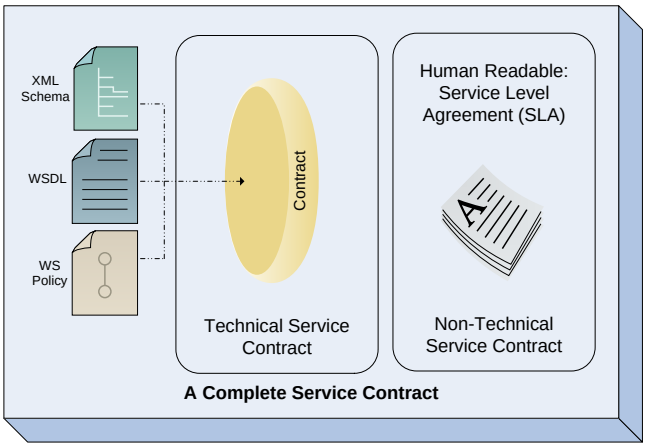


Figure 17: The Service Contract includes the Technical Service Contract, plus the human readable Service Level Agreement (SLA); all parts of a Service Contract are affected by this principle.

Often, the quality of the initial release of a service contract determines the longevity of a service. The sooner the contract needs to be modified, especially in a non-backward-compatible way, the shorter the life of the service will prove to be. With the need of versioning comes the challenge of service and contract governance (Mukhtar, 12/2011).

2.1.3. Goals

A meaningful application of this principle allows for:

- Easy and intuitive understanding of the capabilities of a service
- Reduced need for data model transformation
- Reduced need for data format transformation
- Increased intrinsic interoperability

2.1.4. Types of Service Contract Standardization

There are two aspects of service contract standardization as discussed below (Mukhtar, 12/2011).

- Standardization of Functional Expression
- Standardization of Data Model

2.1.4.1. Standardization of Functional Expression

Functional Expression standardization refers to having each service express details of its functional context using standard conventions. This includes the naming conventions for the services and the service capabilities to comply with an existing enterprise (or domain) standard. For instance, an *Entity* service should be named according to the business entity it models, and a *Task* service should be named based on the business process the service is automating. Similarly, the service capability names should include a verb followed by a noun, and that the service capability names should not repeat their service names.

2.1.4.2. Standardization of Data Model

Data Model standardization usually results in WSDL definitions that share common XML schemas. Once these standardized XML schemas define the I/O for each service capability, the need for data model transformation is naturally reduced, resulting in relatively simple and more efficient service activity. The goal of data transformation avoidance is thus materialized through the standardization of data representation across service contracts as illustrated in Figure 18.

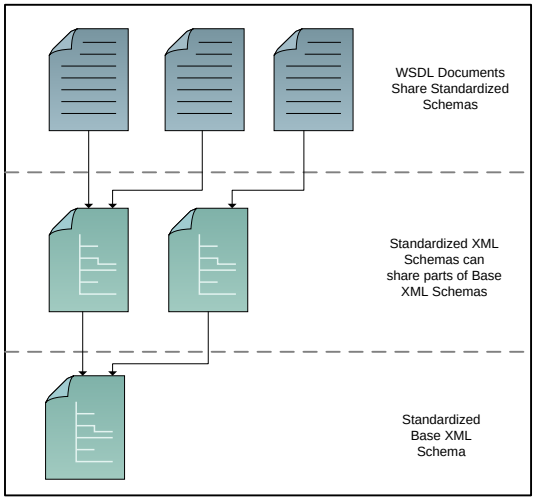


Figure 18: Schema Centralization - the functional context of a service often includes multiple data structures. This overlapping of data-boundaries provide opportunity for Schema Centralization, which means expressing complex data structures in a standard format across disparate applications. That way, when the services need to share data while participating in a Composition, no (or minimal) data model transformation is necessary (Mukhtar, 12/2011).

2.1.5. Service Level Agreements – Non-Technical Part of a Service Contract

During the initial stages of an SOA adoption, organizations tend to focus on the technical part of the service contract that includes standardized data models based on XML schemas, and the functional expressions based on WSDL. The Quality of Service (QoS) related aspects that are expressed in human readable form through the SLAs are often ignored at this initial adoption stage. The SLA part of the service contract, though not technically binding, is often legally binding, and thus should be considered an important extension and integral part of the complete service contract. Below is a small example list of the items that are often included in an SLA. Because of the very nature of this part being human readable, business expectations might be firmly set and measured against the actual service performance (Mukhtar, 12/2011).

- Response-time guaranty from individual service capabilities
- Average response-time guaranty from service compositions

- Availability guaranty
- Protocols guaranty for any scheduled and unscheduled downtime
- Availability of the service utilization data and statistics
- User feedback protocols and guaranty of the feedback review

2.2. Service Loose Coupling

2.2.1. Historical Underpinning in Conventional Software Design

Loose coupling as a software engineering best-practice is well recognized even in the conventional software design paradigms. In early 70s Parnas & Morris explained how modularity is connected with independence, and presented these as a mechanism for improving the flexibility and comprehensibility of a system. Commenting on the idea of product flexibility, they further state that "it should be possible to make drastic changes to the module without a need to change others" (Parnas & Morris, 1972). Taking this idea further, the Layered Architecture is also well-established for its utility in the traditional software architectures. In that pattern, loose coupling is achieved by limiting the visibility of the modules to within their layer only, for instance, as Hunter reports and urges the "separation of data access from the business logic" (Hunter, October 11 2008). Inspiration for the principle of Service Loose Coupling in SOA can be directly traced back from such preexisting notions.

2.2.2. Definition and Discussion

"Service contracts impose low consumer coupling requirements and are themselves decoupled from the surrounding environment" (Erl, 2008, p. 167).

The term ‘coupling’ means the connection or relationship, and the resulting dependency, between two software programs or other IT elements (Kannan, Bhamidipaty, & Narendra, 2011). These relationships are directed to both internal and external elements of a service, and can be measured via some dependency metrics (Karhikeyan & Geetha, 25-27 April 2012). While the internal elements comprise of the program logic that a contract encapsulates, the external elements are the consumers of a service. In the context of service orientation, coupling specifically implies these dependency relationships around a service contract. The principle of loose coupling advocates minimizing negative forms of coupling in order to increase the independence of service contract from its underlying implementation as well as from other consumers.

Although, it is more of a regulatory principle in that it enables other principles to achieve their specific target characteristics, principle of service loose coupling does specifically target achieving a functional context that is as independent as possible of the outside logic. The Service Autonomy is the other design principle that this principles directly impacts as discussed later. Increased positive coupling results in increased design-time control of a service (Mukhtar, 12/2011).

In order for loose-coupling to enables service-reuse, strong coherency must be maintained. However, when new service compositions are created from loosely coupled services that are independent (i.e. owned by different parts of the organization, based on disparate technology assumptions, and evolving on independent schedules and with diverse priorities) the coherency of the composite application can be a challenge (High, Krishnan, & Sanchez, 2008). In any distributed application architecture, coupling is a natural and unavoidable phenomenon that must be well understood. The architectural

adjustments to the extent and direction of such coupling further the goals of Service-Orientation. The application of the service loose coupling principle could result in some additional runtime processing compared to what two tightly coupled services might require. The price of the independence usually manifests in the form of increased data-exchange and data-processing since each service chooses not to depend on the other. On the flip side, however, this price is considered worth the target-state, which is an environment in which services, along with their consumers, can evolve independently over time, with minimal impact on each other. The traditional *point-to-point* architecture – the antithesis of this principle – "is not scalable and very complex as the number of integration points increases as the number of systems increases and can quickly become unmanageable" (Papazoglou & van den Heuvel, 2007). Since service contract and service logic both can form dependencies on parts of the service environment and on each other, the scope of this principle encompasses both the logic of the service and the design of the service contract. As Zhang and Zhou has proposed, the degree of loosely coupling measurement for individual services can be quantified, and can be analyzed from the service dependencies among its disparate consumers (Zhou & Zhang, 2009).

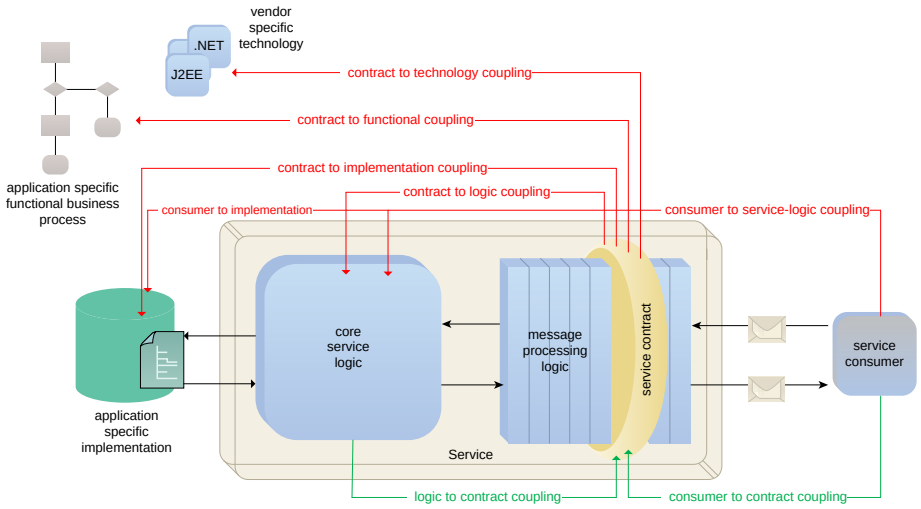


Figure 19: Coupling Types. There are several distinct types of dependencies (or couplings) that a service can have. Loose coupling principle promotes independence in the design of service contracts and allows for free evolution of service logic. It's important to understand the governance implications of such coupling types. More the negative coupling types find their way into the service contract, greater the governance burden and maintenance overhead of a service will be. (Note: negative type couplings are shown in red arrows; positive couplings are depicted with green arrows.) (Mukhtar, 12/2011)

2.2.3. Positive and Negative Coupling Types

There are several types of coupling that relate to either internal or external service design and runtime service activity. These different forms of coupling represent dependencies that exist between the three distinct architectural elements, i.e. Services, Messages and Contracts, but most often, the last. As illustrated in the Figure 19, there are two positive coupling types, and five negative coupling types. Positive coupling types are desirable couplings that are sought by SOA architects, whereas the negative coupling types are undesirable. Much of the effort in Service-Orientation goes into designing services that avoid the negative types of coupling to the extent possible. Below, these coupling types are discussed individually (Mukhtar, 12/2011).

2.2.3.1. Positive Coupling Types

- Logic to Contract

- Consumer to Contract

2.2.3.1.1. Logic to Contract Coupling Type

If the service contract is written first, (following the *Contract First* approach as discussed earlier), the implementation that follows, in the form of service logic, gets tightly coupled with the existing contract. This is a desirable outcome. It means that in the future, if the service implementation needs to change, the service contract would remain impervious to it, and so will the service consumers.

2.2.3.1.2. Consumer to Contract Coupling Type

When a consumer binds to a service contract, the resultant relationship is called a consumer-to-contract coupling. It is a desirable form of coupling because it achieves most independence between the consumer and the service logic. Service-Orientation advocates that all communication between a service and its consumer occur via the published service contract. If this contract is designed independent of the service implementation, all consumer coupling will be limited to this published contract, leaving the service implementation logic and the consumer decoupled.

2.2.3.2. Negative Coupling Types

There are five types of negative couplings that must be closely watched during the service design stage. The desired target-state is to avoid, or minimize, such undesirable couplings in order to maximize the governance independence of the service contracts. These undesirable couplings types are listed below and illustrated by Figure 19 in red arrows.

a. Contract to logic

b. Contract to functional

c. Contract to implementation

d. Contract to technology

e. Consumer to service logic and/or implementation

2.2.3.2.1. Contract to Logic

Contract to logic is the negative coupling type that is most associated with the bottom-up approach of service development. If the service implementation already exists and the service contract is derived from it using auto-generation tools, the resulting contract will be closely tied to the underlying implementation. Due to this inheritance effect, in the future, when the implementation needs to change, the contract will have to change with it. In such a scenario, service logic will not be dependent on the service contract, instead, with a new or modified implementation, a new or modified contract would need to be regenerated. Furthermore, with a modified service contract the service consumers would need to be changed as well. This, of course, is the antitheses to the promised target-state of Service-Orientation.

2.2.3.2.2. Contract to Functional

If a service contract has specifically been designed and developed in support of a pre-existing business-process, an existing consumer, and/or a task service, it might result in tight coupling of the service contract to the underlying process or to the specific consumer. As the internal implementation of this underlying process changes, the service contract will have to change with it. Same dynamic can happen when the contract is specifically developed for an existing consumer. In a B2B implementation, for example,

if consumer-specific implementation details are allowed to seep into a service contract, that contract will probably be forced to change when the service consumer is modified.

2.2.3.2.3. Contract to Implementation

Service logic can potentially envelope several implementation specific elements like legacy APIs, vendor specific database functions, physical server environments, network specific paths, file names and user account information. Program logic can have direct or indirect dependencies on such implementation elements, but if these dependencies spillover the service contract, that can form the negative coupling type called contract to implementation coupling. Given the fact that these implementation specific details do tend to change frequently, contract to implementation form of coupling is one of the nastiest one, and easy to be overlooked by a service designer.

2.2.3.2.4. Contract to Technology

If the service contract itself is not technology agnostic, it can be said that contract to technology coupling exists. It is a form of negative coupling in that it limits the consumers of your service to use only the underlying technology. The consumers that cannot use that specific technology are left out. Web Services offer a technology agnostic way of creating service contracts, but mere use of WSDL does not bulletproof a contract from this negative type of coupling. A WSDL can include technology specific data types, say .NET specific calendar field, and hence still end up with contract to technology coupling. The point to keep in mind is not to include anything in the service contract that might be specific to the underling implementation technology.

However, there's a silver lining to this negative coupling type as well. Since making the contract technology agnostic usually involves transforming data into a standard format, like XML/SOAP, technology specific communication is much faster. Marshaling and un-marshaling is the necessary price that is to be paid to keep a contract technology independent. In certain cases, however, when all (or an overwhelming percentage of) consumers of a service are developed in one technology, it might be worthwhile to consider allowing this coupling. In case not all but most consumers of a service depend on one technology, a service may be exposed via two contracts – applying Concurrent Contracts design pattern – one in the target technology and the other, for the small minority consumers, in a technology agnostic form. A tradeoff price in terms of governance overhead is naturally to be paid with such a configuration.

2.2.3.2.5. Consumer to Service Logic and/or Implementation

Since one cannot always force a potential consumer to necessarily access a service or its underlying resources through a published service contract, it is possible of a consumer to simply bypass the service contract and to connect directly to the core service logic, or even to the underlying resources of a service like a database. This is one of the worst forms of coupling as it tends to defeat the purpose of Service-Orientation. From maintainability perspective also, it represents worst case scenario. Business implementation is bound to change with time, and when it does, the consumer would need to be modified as well. Implementing the Contract Centralization design pattern could provide a solution, which requires that all service consumers interface with a service exclusively via the officially published service contract, and not through other potentially available resource entry points.

2.2.3.3. Percolation of Negative Coupling – the Unintended Inheritance Effect

The negative couplings, as discussed above, could create more serious and far-reaching problems that might not be readily obvious to a casual eye. Building consumer-to-contract coupling is a recommended approach which helps avoid negative coupling types. However, if the service contract is poorly designed, consumer-to-contract coupling can also produce unintended negative couplings. Because of the cascading effect, such negative type of couplings could percolate downstream and unintentionally form tight coupling with deeper architectural element.

As discussed in the Positive Coupling Types section previously, service consumers are expected to develop the positive form of coupling to the service contract. Any of the four negative coupling types carried by a contract, however, can produce a domino effect. If this happens, all subsequent service consumer programs will end up forming the same dependencies to the underlying implementations. Worst, the consumer designers will have least idea that their programs are involuntarily getting tied to something beyond the published service contract, seamlessly inheriting the negative coupling from it. These types of indirect coupling can lead to serious flaws in the overall design of an entire solution, and result in far-reaching and expensive reworks and modifications later in time.

Thus, the undesirable forms of coupling allowed into the service contract design eventually ends up being imposed upon, and proliferated through, all consumers of the service. Specially, in the case of agnostic services that need to be highly reused, the problem can be magnified exponentially, and must be watched carefully at the original service design stage (Mukhtar, 12/2011).

2.3. Service Abstraction

2.3.1. Historical Underpinning in Conventional Software Design

Writing on the subject of modularization in 1972 Parnas & Morris instructed us to hide data from the objects that do not have "a need to know" (Parnas & Morris, 1972). The rationale behind this wisdom being: "if the data structure is changed, the other objects do not have to be notified about the change" (Bahill & Botta, 2008). In 2000, Gomaa reiterated this principle, and called it "information hiding and function hiding" (Gomaa, 2000). Abstraction and information hiding form the foundation of all object-oriented development. As Booch explained, large object-oriented systems tend to be built in layers of abstraction, where each layer denotes a collection of objects with restricted visibility to other layers called subsystems (Booch, 1986). In SOA, the principle of Service Abstraction directly traces back to this well understood notion in conventional software design. However, SOA simply shifts the fulcrum from Object to Service, and heeds to the advice of privacy experts to disclose the minimum set of information needed to complete a transaction (Breaux, 2014).

2.3.2. Definition and Discussion

The principle of Service Abstraction may be defined as "Service contracts only contain essential information and information about services is limited to what is published in service contracts" (Erl, 2008, p. 214).

At the fundamental level, this principle advocates deliberate hiding of service metadata such that only necessary and minimal information is available to the service consumers, and that too, only via published service contract. All other non-essential

information about the service and its capabilities is abstracted away from the consumers and consumer designers.

As with service Loose Coupling principle, instead of incorporating any specific target characteristic, this principle is a regulatory principle which supports and enables other principles. Most directly, this principle influences the extent of loose coupling that can be attained, and the design of service contract. The application of this principle also influences the service reusability, service composability and service discoverability principles by regulating and limiting the amount and nature of the metadata available for the service consumers (Mukhtar, 12/2011).

The idea of information hiding is not new and is fairly straightforward on the surface. It's borrowed from Object Oriented design world where it's known as the principle of encapsulation – limiting access to an object's data members only through the *accessor* (get/set) methods. In Service-Orientation, however, the tricky question is: how much information hiding? Achieving the proper balance in such information-hiding is this principle's core idea. Too much information hiding can impede the consumer's ability to utilize the service effectively; on the other hand, too little information hiding can encourage the consumers to develop unwanted dependencies, and undo loose coupling.

As more information is published in a technical service contract, tighter the subsequent consumer-to-contract coupling becomes. Consumer-to-contract coupling is a positive form of coupling which is a desirable characteristic of SOA target-state. However, if the consumer-designer has access to the information regarding the actual implementation of the service, and the service contract is liberal enough to allow using

such privileged information, the undesirable type of consumer-to-implementation coupling can still seep in. This subtle point can be elaborated with the following example. Suppose that a service X is implemented using .NET Components technology that uses Security Tokens and Active Directory references. Also, suppose that the service X contract-designer has intentionally reduced constraint granularity in order to achieve high service reusability. Given this possible scenario, a consumer-designer, with this privileged information regarding internal implementation detail of the service can choose, in good faith, to utilize it by directly passing into service X, security tokens and hard directory references, in order to gain a performance advantage. This, of course, will work and probably even produce better performance for this particular consumer, until suddenly one day it will stop working all together, when, for some reason, the underlying service implementation gets modified.

Similarly, as more information is made available to the consumer-designers in the form of human readable non-technical part of the service contract (i.e. SLAs), greater will be the potential that the designers will base their consumer programs on this additional-information, and unintentionally end up binding to the service too tightly.

Thus, increasing the consumer awareness about all aspects of the service is not necessarily a good thing in all cases. The Service Abstraction principle asks the service designer to take the time and assess and balance, both risk and value propositions of publishing specific pieces of Meta information about the service (Mukhtar, 12/2011).

2.3.3. What to Hide and What Not to Hide is the Question

When determining what service information one must hide and what information one must expose in the service contract, it is often helpful to partition all available information in the following four distinct categories (Erl, 2008, p. 218).

a. Functional metadata

b. Quality of Service (QoS) metadata

c. Implementation Technology metadata

d. Programmatic metadata

The proper implementation of this principle turns a service into a black-box that consumers know how to use, but don't know how it works. This abstraction and black-box concept permeates the commercial software designs because the commercial software vendors want their customers to be able to use their products without giving out any information on how those products actually work. Service-Orientation treats the services in the similar vein; abstraction within the enterprise thus becomes a serious design consideration. The consideration for each of the Meta information categories listed above is discussed below (Figure 20) (Mukhtar, 12/2011).

Figure 20: The four common Meta information types. Each describes the service from a unique perspective. Meta information related to Programmatic and Technology aspects of a service generally are of more

concern for Abstraction than Functional and QoS related information. (Mukhtar, 12/2011)

As with the other service design principles, the question is never all or none; whether a service design should or should not abstract, but how much should be abstracted. In each of the Meta categories discussed below, the service design architect needs to make a judgment decision as to where the proper balance rests.

2.3.3.1. Functional Metadata

Within a functional context, services often have more capabilities at the fine-grained level than are exposed via its technical contract. The question for the architect designing the service is which fine-grained capabilities should be exposed and which should be abstracted out. For example, assume service S contains capabilities A, B and C. Also assume that capabilities B and C provide partial functionalities which capability A aggregates by composition. Should capability B and C be exposed in the service contract or should only capability A be exposed? In order to answer this question the service architect needs to figure out if there are other possible direct consumers of capability B and C. If not, the capabilities B and C are better left abstracted. Service contract is mainly the region impacted by functional metadata abstraction.

2.3.3.2. Quality of Service (QoS) Metadata

A wide range of non-functional, utilization, reliability and behavior related information which is usually part of the SLA of a service is accounted under QoS metadata. Alternately, if the deployment environment supports, these SLAs could also be implemented using WS-Policy Assertions. Either way, most of this design-time

information should be shared with the consumer-designers in order to set their expectations. However, there's no need for service to share any runtime QoS metadata.

For instance, peak-time business hours, say "9AM to 3PM – Mon to Fri", and performance expectation during this time, say "a response within 3 seconds", should be shared with the consumer-designer so that they can realistically set their program parameters, say "timeout limit during the day". However, runtime information, for example, the current state of service X on node 3, cluster 5 of the application container, does not need to be shared. Service contract is mainly the region impacted by QoS metadata abstraction, but because of the behavior and reliability related details, core service logic region can also be impacted.

2.3.3.3. Implementation Technology Metadata

As the name suggests, this type of information includes metadata related to the technical implementation of the underlying service logic. Some of this information is needed by the consumer to be able to use a service, but there's other information that the user does not care about and should not know. Any commercial off the shelf (COTS) software program can be a point in case – as a consumer of such a program one needs to know how to install it and how to execute it, what are the available interfaces, and with which operating systems the program will work. However, one does not need to know which programming language the program code was written in, and which particular version of the compiler the program was compiled with. Not only that a consumer does not need this kind of information, but in fact, this kind of information in the wrong hands can be harmful to the program and its vendor. Naturally, the COTS vendor wants to protect his program by abstracting this kind of meta information. During service design, a

service architect needs to take the same approach towards sharing implementation technology related metadata of a service. In Service-Orientation, this technology related metadata abstraction not only protects a service but also protects the consumers. When the consumer designer doesn't know the underlying implementation technology details, there is less of a chance that the consumer programs will develop unintentional tight coupling to a service.

For instance, in case of the service implementation via web services technology, information that the service can be invoked using SOAP 1.1 is necessary to be shared with the consumer, but information that the service was developed using Java 1.5 is unnecessary and undesirable. Core service logic as well as message processing logic can be impacted by implementation technology metadata abstraction.

2.3.3.4. Programmatic Metadata

Information regarding the low-level program logic and routines include things like exception handling data, specific computational algorithms, authentication and authorization programming logic and logging related programming details. Consumers of a service do not need to know these programmatic details of a service. However, some organizations nurture open IT environment and make available even the program source-code to anyone interested. Open Source projects, and some organizations using these projects are good examples. This can make the programmatic metadata abstraction challenging, and may require an organizational and cultural change.

The service abstraction principle advocates hiding of programmatic metadata from the consumer-designers in order to increase the longevity of the consumer programs and decrease the governance burden for the service. It also provides the service the freedom

to evolve freely without any constrain from its consumers. By abstracting away this level of information, an architect avoids inadvertent consumer-to-logic negative coupling. Core service logic is mainly the region impacted by programmatic metadata abstraction.

2.3.4. Organizational Impact

Proper implementation of this principle has the potential not only to change the IT department but also how an organization works as a whole.

Within the project, it introduces "hidden composition" issue that can impact service performance expectations and what is committed to in the SLAs. Because this principle advocates limiting the information about a service to its published contract, it is very much possible that what a consumer is invoking as a service is in fact a service composition controller, i.e. a service capability at the top of a service composition hierarchy, composing other services. In this common scenario, some inter-service performance computation will be required to set consumer expectations in terms of service performance. For example, assume consumer C invokes service X, which in turn composes capabilities in service Y and service Z. In order to satisfy service X's SLA of say, "responds within 2 seconds", service Y and service Z must not take more than 2 seconds of processing time collectively. From consumer C's perspective, services Y and Z do not exist; he's only aware of the promise made to him in service X's SLA. However, the designers of service X are aware of Y and Z as participating members in their composition. They must keep in mind limitations of service Y and service Z when writing service X's SLA.

Application of the service abstraction principle may also involve the need to introduce or tighten the access controls for service design documents as well as its source

code. As discussed above, even within the IT department, other teams need to be denied access to some of the service related meta information. This measure protects both service as well as its consumer from developing undesirable negative types of coupling. However, this implies changing the organizational culture of information sharing across project teams, and might require some explanation and justification at the enterprise level.

The impact of implementing this principle can, thus, go beyond current project and program, and can require changes at the organizational and cultural level of the whole enterprise. For the sake of the service abstraction principle, a detailed formalization of the processing resources are lacking for the individual service capabilities (Stachtiari, Vesyropoulos, Kourouleas, Georgiadis, & Katsaros, 2014).

2.4. Service Reusability

2.4.1. Historical Underpinning in Conventional Software Design

The Single Responsibility Principle (SRP) from the traditional software architecture and design paradigms is well known and understood. Originated by Martin and quoted by others, it states that "every object in a system should have a single responsibility, and all the object's services should be focused on carrying out that single responsibility" (Haoyu & Haili, August 2012; Martin, 2003). Thus, reuse is a well-regarded concept in traditional software development, but it is merely a convenience, whereas reuse is essential in the case of services, because services cut across organizational boundaries (Huhns & Singh, 2005).

2.4.2. Definition and Discussion

The principle of Service Reuse may be defined as "Services contain and express agnostic logic that can be positioned as reusable enterprise assets" (Erl, 2008, p. 259).

Reusability is one of the most fundamental principles of service design which influences all other principles (Welke, Hirschheim, & Schwarz, 2011). It seeks to increase the potential of a service to be reused by the consumers beyond the original requirement for which it was designed and developed. This principle advocates actively looking for potentially reusable unit of logic and making it available within an agnostic service context. Service-Orientation design paradigm assigns unprecedented weight to the principle of reusability, bring it on par with the commercial software engineering practice. In fact, this principle combines techniques and methodologies from traditional commercial software design to that of silo-based enterprise product design and development. This stress on reusability clearly points to the fact that commercial product design is one of the major influences on Service-Orientation design paradigm.

Service reusability principle promotes several of the strategic goals of Service-Orientation and helps develop some tangible target characteristics. By focusing on agnostic service context, this principle directly supports increased ROI, increased organizational agility and eventually, reduced IT burden (Mukhtar, 12/2011).

A service can be considered agnostic if its capabilities expose functionality that is not tied to a particular business process, but instead, exposes generic multi-purpose logic that can be used by several different business processes. As a general rule, more generic the encapsulated solution logic, higher will be the reusability potential.

In Service-Orientation design paradigm, reusability represents a core target-state characteristic that directly ties to the goal of increased ROI. This principle sets the stage for materializing this goal by actively seeking reusable units of logic and building agnostic services. To be specific, Logic Centralization and Contract Centralization are the two vehicles with witch Service-Orientation achieve service reusability. The result is a highly standardized and normalized service inventory maximizing reusability (Mukhtar, 12/2011).

2.4.2.1. Logic Centralization Pattern

Logic Centralization pattern limits the number of implementations of a particular business-solution logic. It advocates that at any given time, there should be one and only one way of executing certain solution logic – technically known as Service Normalization which emphasizes service boundary alignment.

This puts the onus on the architects and designers of a solution not to rebuild something that already exists elsewhere. On the other hand, the responsibility is also shared by the owners of the existing solution logic to make it available, accessible and useable by project teams, other than their own. Redundancy of solution logic is undesirable in SOA design[3]. In such an adverse scenario, one either must take on the governance responsibility of keeping the logic in sync at two different places, or accepting potentially inconsistent and irreconcilable results. If service A encapsulates business logic to do X, all subsequent services should form effective compositions with service A to execute solution logic X. All efforts should be spent not to inadvertently (or

[3] Jeff Bezos (Amazon's CEO) issued a stern warning in 2003 to all the internal software teams that each service within the company must be consumed only by a single well-documented contract. This famous edict is often credited by the experts for the tremendous success Amazon IT achieved in SOA adoption.

otherwise) redevelop logic to perform X in another service capability. This design pattern applies to all services, but especially to the agnostic services because of their high potential of reuse.

The ideal implementation of the Logic Centralization pattern can prove challenging at the enterprise level, especially in large organizations. However, a reasonable extent of application of this principle could involve implementing it in conjunction with the Domain Inventory design pattern, which advocates segmenting the organization in manageable business domains (Mukhtar, 12/2011).

2.4.2.2. Contract Centralization Pattern

Logic centralization pattern, as discussed above, addresses only part of the problem. It tackles the issue of single vs. multiple copies of solution logic residing within a service inventory, and advocates that there should be only one copy. It does not, however, address the question of how that solution logic should be invoked – that is the Contract Centralization part. Contract Centralization pattern demands to limit the access to a service only through its published contract. This means that no consumer should be build such that can connect and execute the encapsulated logic of a service, bypassing the published service contract (Figure 21) (Mukhtar, 12/2011).

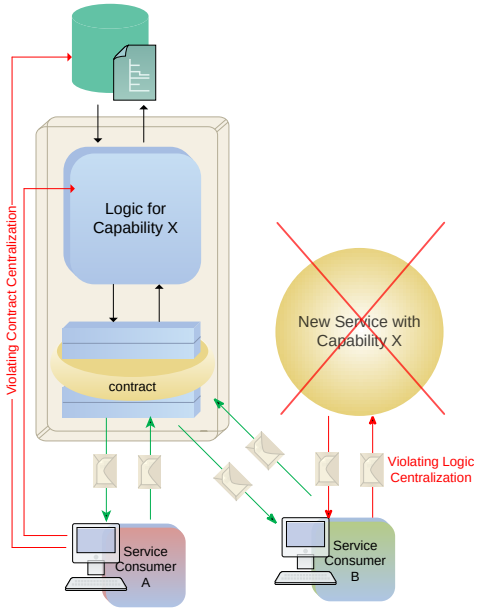


Figure 21: Applying Logic Centralization and Contract Centralization design patterns together establish a single official entry-point to a normalized service. (Mukhtar, 12/2011)

2.4.3. Types of Reuse

A service can be reused in two different ways: by being repeatedly invoked from the same consumer for the same business task, or by being invoked by different consumers for different business processes. Although, both types of reuse increase ROI, it's the second type of the reuse where the real prize lies. By designing highly generic services with agnostic functional context the principle of reusability tries to maximize reuse from different consumers resulting in a highly composable service inventory. The Principle of Service Composability (discussed below) directly supports this goal (Mukhtar, 12/2011).

2.4.4. Considerations for Creating Agnostic Services

Emphasis on reusability implies some stringent design considerations that affect all parts of the service. Service-orientation "should apply equally at all levels in the business such that there is no distinction between larger-grained 'business services' and finer-grained 'IT services'" (Nayak & Nigam, 23-26 July 2007). Below are some guidelines for creating agnostic services that can be leveraged multiple times and can survive for a reasonable period of time (Mukhtar, 12/2011).

- The upfront time spent on defining service inventory blueprint provides great opportunity to identify agnostic units-of-logic and to fashion them into agnostic services. Sometimes, pressure to meet delivery deadlines on projects can eclipse the centrality of this part of the process. When possible, don't rush through this process.
- Decompose the high-level base-logic of a business requirement into such reasonable size units-of-logic that can maximize reusability, but not too granular that the overhead of gluing them together would overweigh the potential benefit. Incorrect granularity could mean that a service covers too much functionality or too little functionality. A key challenge that SOA architects face is to determine the most appropriate level of service granularity, which could be quantitatively measured depending upon the granularity attributes, i.e. reusability, composability, complexity, business value, and context-independency (Khoshkbarforoushha, Tabein, Jamshidi, & Shams, 5-10 July 2010).
- Seek active involvement of business domain SMEs and service analysts to ensure that the service-context boundary and contract granularity represent true

functional context. The service analysts, who are well familiar with the detailed contract requirements, might be tempted to define an extremely fine-grained service contract. Passing extremely sanitized I/O parameters to your service makes sense if you're only thinking of your current project requirements. However, a broader vision of reusability potential might suggest a lenient and coarse-grained contract to keep room for other consumer to join in later.

- No matter how well the Logic Centralization and Contract Centralization design patterns are followed, changes are bound to occur. A well designed and regularized versioning system will help during the service evolution stage.
- Finally, if agnostic service context means reusability, it also means the need for scalability. As the amount of reusable services grow, so does the need for a highly scalable runtime environment where agnostic services can be deployed as effective compositions. Stress-testing such a deployment environment for peak usage is an advisable strategy.

2.5. Service Autonomy

2.5.1. Definition and Discussion

The principle of Service Autonomy may be defined as "Services exercise a high level of control over their underlying runtime execution environment" (Erl, 2008, p. 296).

The principle of service autonomy advocates that the services should have maximum amount of control possible over their underlying resources and environment. It asks the service architects to carefully consider all direct and indirect dependencies that the service will form, and the potential performance impact of such dependencies. Naturally,

more independence and isolation a service enjoys in terms of its underlying resources, more predictable its performance will be. On the other hand, more resources a service shares with other programs, less predictable and reliable its performance can expected to be.

Reliability, consistency, and behavioral predictability are the three specific target-state characteristics sought by the application of this principle. The principle of autonomy also directly supports the reusability and composability principles (Mukhtar, 12/2011).

Autonomy refers to the ability to self-govern, i.e. the freedom and control to make internal decisions without dependency over external elements (Kannan, Bhamidipaty, & Narendra, 2011). In relation to Service-Orientation, a service is said to be autonomous if it is able to carry out its logic independently without getting influenced by external factors.

With the understanding that in most environments, complete service autonomy is practically not possible, this principle urges an architect to consider and realize any opportunities to isolate the underlying resources of a service, and make a service as self-sufficient as possible. On the flip side, achieving high service autonomy can significantly increase infrastructure requirements and costs.

The most common shared IT resources are databases and server infrastructure. Because the database entities usually cut across the business domains and functional contexts of services, it is most probable that a service will have to share database resources with other programs and services. The concurrency of use in such cases impacts the consistency of service performance. A high utilization time, for example, of another application or service that is sharing a database with a service can negatively

impact its performance without much advance knowledge. Similarly, it's naturally difficult to meet promised performance related SLAs consistently, if the server that is hosting a service is also hosting ten other services with irregular peak utilization hours (Sud, 2010) (Mukhtar, 12/2011).

2.5.2. Service Autonomy vs. Service Composability

In a way, the considerations for the Principle of Autonomy compete with the considerations of the principle of Service Composability. A service naturally tends to diminish its autonomy as it moves up the hierarchy of service composition tree. Also, due to the principle of Service Abstraction, a consumer-service designer might not know anything about the relative position of a provider-service within a service composition hierarchy i.e., what is invoked as a service might in fact be a large and complex composite service. For example, assume that capability C1 in service S1 invokes capability C2 in service S2. Even if S2 is not a composite service, S1 has relented a certain amount of control by delegating work to C2, which means S1 is now dependent over S2. Furthermore, the fact that S1 does not know where S2 stands in a possible service composition hierarchy makes it even more difficult for S1 to measure its own level of autonomy. Thus, in this case, the autonomy of S1 would be reduced by the total autonomy of S2 composition.

This dynamic of service aggregation into service compositions is a natural outcome of Service-Orientation which cannot be avoided, but by understanding it better, an architect can make educated and balanced design decisions. An SOA architect needs to find the best equilibrium between these two competing considerations, and tailor the solution according to the project limitations and ground realities (Mukhtar, 12/2011).

2.5.3. Types of Autonomy

There are two distinct types of autonomies that this principle refers to as listed below (Mukhtar, 12/2011).

a. Design-time autonomy

b. Runtime autonomy

As a general rule, more design-time autonomy translates to greater potential for the runtime autonomy.

2.5.3.1. Design-time Autonomy

The level of freedom service designers have to modify their service in the face of external dependencies is referred to as design-time autonomy. Once a service contract is published, service consumers inevitably develop dependencies on it. As explained earlier, this is referred to as consumer-to-contract coupling which is a positive type of coupling. We, as service designers, seek to maximize consumer-to-contract coupling because it limits consumer dependency to the service, leaving the service implementation and core logic to freely evolve. However, if the service contract is poorly designed or the contract is itself based on and is derived from some existing implementation (bottom-up approach), the technology and implementation related features can creep in and percolate up to the service consumer, right through the service contract.

For example, assume that a project team, under pressure to show results of SOA adoption, generated service contract SC using automated tools from an existing legacy implementation encapsulated by service S. In this bottom-up development approach, also assume that the legacy implementation included a vendor technology specific data-structure that the WSDL generation tool directly translated into XML schema complex-

type. By exposing this underlying technology specific feature in SC, the project team has inadvertently limited their design-time autonomy. Any future change to this data-structure would inevitably trigger an accompanying change in the service consumer as well.

2.5.3.2. Runtime Autonomy

The level of control a service has over its execution environment and underlying resources when the service is invoked and is running, is referred to as runtime autonomy. It's an important design consideration because it enables a service designer to commit to specific runtime performance guaranties to the service consumers. These guaranties could be published in the form of SLAs, as expected performance matrix (for example, by time of the day and response time), or, these guaranties could be related to performance reliability and security expectations.

It makes it increasingly difficult to provide the above mentioned performance guaranties if the service is dependent upon, and/or shares runtime environment with, other programs and services. Increased physical service isolation translates into greater service runtime autonomy and vice versa. Since agnostic services that model into Entity and Utility services has greatest potential of reuse, these services end up being part of complex compositions. Even though this is a desirable dynamic, on the flip side, providing performance guaranties for such complex composition becomes challenging at best.

2.6. Service Statelessness

2.6.1. Definition and Discussion

The principle of Service Statelessness may be defined as "Services minimize resource consumption by deferring the management of state information when necessary" (Erl, 2008, p. 331).

Digging deep in to the principle of Service Statelessness, Atkinson & Bostan makes a point of distinguishing between the "Observable" and the "Inherent" state. These researchers aptly point out that only the latter distinguishes whether the responsibility for storing the state is "internal" or "external" to the implementation of the service. They further affirm that the principle of Service Statelessness is about this Inherent state (Atkinson & Bostan, 2009).

During its execution, a service transitions through several stages of processing activity. During these stages, the service is not always actively working on all the data that has been passed to it or that it has generated from its own processing. Holding and managing excessive context related state data in working memory negatively impacts the service performance, especially if it's a large Task business service running in a choreography engine like BPEL (Jain & Kumar, 2007). A service should therefore be ideally designed to hold only the necessary state data that it's currently working upon. All other state data should be tucked in somewhere it can be retrieved quickly and efficiently when needed while the other processing continues.

The name of this principle is somewhat misleading. Service statelessness principle does not advocate that building stateless services, but it does ask an architect to consider and avail the opportunities to defer and delegate the state related data whenever possible.

Thus, the service statelessness principle aims to establish mechanisms to support runtime deferral of state data in order to minimize stateful condition while the service is active.

Increased availability and scalability are the specific target-state characteristics sought by the application of this principle (Mukhtar, 12/2011).

The principle of service statelessness is all about increasing service availability, scalability and improving the service execution performance. This principle is somewhat different from the rest of the service design principles in that it seeks to modify the core service logic temporarily. Depending on the approach used to apply this principle and the model of the service under question, different design characteristics can be supported.

All programs, including services, are required to temporarily hold data related to the task under execution. This task-related context data is referred to as state or state data. At times, especially in n-tier applications where numerous clients concurrently invoke a server-side program (a service in Service-Oriented paradigm) many times over, this state data can stack up very quickly. If services are left to amass this data unchecked, the system performance can be severely impacted as the number of clients increase. Services are therefore designed to remain in a stateless condition wherever appropriate, by deferring the state data to a temporary location.

This temporary location can be a system cache that does not involve this service's resources. It can also be a local database or even a message queue. The particular mechanism of this temporary storage is not important as long as it does not share the resources with the service. Since the idea is to off load unnecessary burden to keep the services as lightweight as possible on the underlying service resources, this temporary location must be really fast in accessibility.

The give and take of the state data in this situation would understandably be extremely recurrent, thus this principle, like the principle of service autonomy, focuses less on service contract and much more on the design of the core service logic. Also, because services themselves are only containers of capabilities, the measure of statelessness would differ from capability to capability within a single service (Mukhtar, 12/2011).

2.6.2. Understanding Deferral and Delegation Processes

Deferral and delegation are two related concepts used in the process of unloading of the state data in application of this principle. *Deferral* implies putting off the management of state data, related to the current activity, to a later time, while *Delegation* implies passing down the responsibility of holding the data to an outside agent. This whole process of temporary relocation of the state data outside of the executing program is referred to as *state deferral and delegation*. The intention is to retrieve the data back at a later point in time to finish the task at hand but is not really needed at the very moment. The management of the state data is postpone (hence Defer), and instead of holding this data in memory throughout the processing of the entire task, it is transferred (hence Delegate) to a local database (or elsewhere) for temporary safekeeping (Mukhtar, 12/2011).

2.6.3. State Types and Conditions

There are four basic state types and service conditions that need discussion in relation to the service statelessness principle. These are listed, illustrated (Figure 22) and discussed below (Mukhtar, 12/2011).

a. Passive

b. Active

c. Stateless

d. Stateful

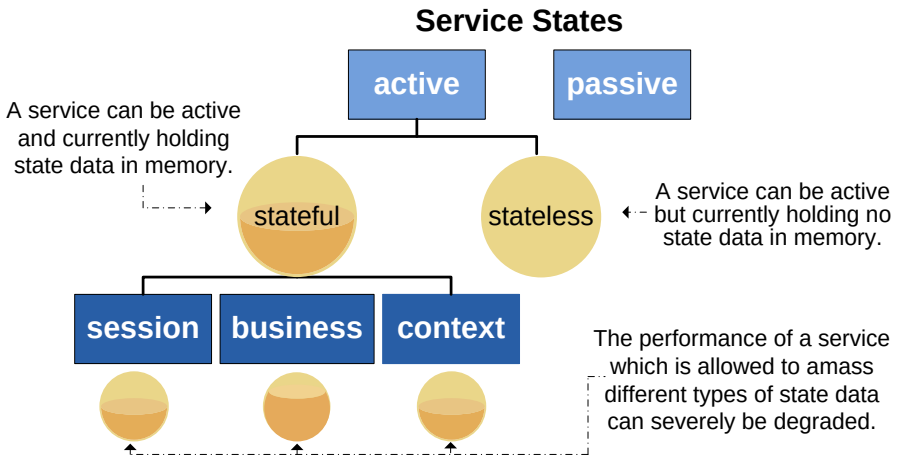


Figure 22: During its execution, a service transitions through several different stages, some requiring it to hold temporary state data. The principle of Service Statelessness asks an architect to adopt a mechanism for the runtime deferral and delegation of such data when possible in order to minimize a service stateful condition. (Mukhtar, 12/2011)

2.6.3.1. Passive State

At a point in time when a service is not yet invoked and is thus not using any underlying resources, it is in a Passive State. In case of component based services, bean/object containers might keep a ready pool of initialized service objects to be served up to the consumers on request. In that case, even though the service might reside in the

object pool of the container in the initialized state, since it has not yet been invoked by a consumer, the service is considered to be in the Passive State.

2.6.3.2. Active State

At a point in time when a service is invoked and is in the process of executing its core logic, it can be said to be in the Active State. In this state, a service instance is constantly consuming a base amount of server memory and CPU cycles.

2.6.3.3. Stateless

A service can be in the Active State but not processing or holding any state data at a particular point in time. It might be waiting for another process to finish, or, for another service that it has composed to gather some data and pass it back to it. Either case, there's no state data to be managed at this point in time. In such a condition, a service can be classified as stateless.

2.6.3.4. Stateful

When a service is in the Active State and is holding and processing state related data, it is called to be Stateful. It can hold some state data while work on gathering some other. Or it can be performing some computations on part of the data while holding the rest for later use. This temporary data can be classified in the following three broad categories.

a. Context related data
b. Business related data
c. Session related data

In any service activity, complex or simple, the invocations between services usually involve a set of data that is passed back and forth between service capabilities. This data can be fine-grained or coarse-grained but is usually more than what is needed for immediate processing – more so when passing coarse-grained than with fine-grained exchanges. Services need this data to set a context of a request that is being worked on. This kind of data is called Context Data.

The business related state data usually include the result-set from database queries. While the core service logic is performing its processing tasks, most of this business data usually need to be held in memory for further processing steps. It is often large and not all of it is needed at once. Thus this kind of state data provide good opportunity to be off loaded until actually needed.

Session level state data is usually tied to a particular user that might need multiple requests to complete a job. Even though the HTTP protocol is stateless – meaning it does not maintain a session beyond current request – webservers and application-servers offer means to remember requestors and thus allow working in a session environment. During the life of such a session, usually some data is generated that need to be kept alive for further processing, but is not needed to be in the memory every microsecond of the session's life. This kind of session-data also provides some possibility of deferring and delegating it out temporarily.

2.7. Service Discoverability

2.7.1. Definition and Discussion

The principle of Service Discoverability may be defined as "Services are supplemented with communicative metadata by which they can be effectively discovered and interpreted" (Erl, 2008, p. 368).

As Atighetchi et al explain, Service Discovery itself is a relatively simple process. A Service registers itself with an existing Service Registry, while a client performs lookup requests on this Service Registry to find newly registered services. Once the client has found a suitable service, it proceeds to invoke that service through a specific invocation mechanism (Atighetchi, Webb, Loyall, & Mayhew, 2010; Hutchinson, et al., Jan.-Feb. 2008). While Discovery is the process of finding services, the Interpretation is the process of understanding the capabilities a service offers. Similarly, Discoverability and Interpretability are the measures to which a service supports the discovery and interpretation processes and thus adheres to this principle.

Increased awareness of available enterprise resources is the specific characteristic sought by the application of this principle.

In a service-oriented organization, "business processes are available [on demand] as services for integration with other business processes [ideally] across the company and with key partners, suppliers, and customers" (Nayak & Nigam, 23-26 July 2007). For the service reusability to work, it is imperative that the services in a service inventory be easily locatable and understandable in terms of their capabilities as well as the data-structures exchanged (Papazoglou & van den Heuvel, 2007). If the services are hidden from their potential users, or are ambiguous as to the functionality they provide, it's

highly probable that someone will waste time and energy in rebuilding what already exists, and defeat the whole idea of service normalization and reusability. Besides service de-normalization, other negative effects include inconsistent results from bloated, convoluted and eventually unmanageable enterprise architecture (Mukhtar, 12/2011).

This principle is all about the quality of communication and effective dissemination of the information about service capabilities through the use of a service profile and/or a service registry. This information includes the content in the service contract as well as the metadata in the corresponding registry/repository record to also describe the nonfunctional aspects (Parlanti, Paganelli, & Giuli, June 2011). In other words, the metadata related to discoverability can be incorporated directly into the technical service contract (e.g. WSDL) in the form of human-readable annotations which does not affect the contract but only explains it. Similarly, discoverability related metadata can also be applied via the use of creative policy assertions by implementing WS-Policy standard (Figure 23). For this metadata to be effectively used by disparate teams that might need to reuse a service, the service profile needs to be readily available and understandable. Unfortunately, even the services provided by ecommerce giants like Amazon and PayPal have sometimes been noted to confuse the service consumers due to ambiguous data interactions and hidden business rules (Saleh, Kulczycki, & Blake, Sep/Oct 2009).

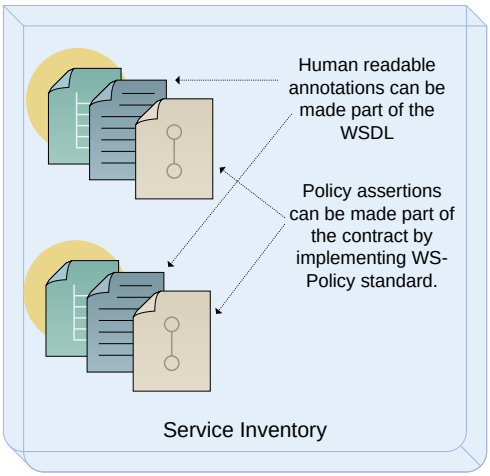


Figure 23: Besides being standardized, all service contracts within a service inventory are supplemented, in a standard way, with metadata that helps in finding and understanding services and their capabilities. (Mukhtar, 12/2011)

Service discoverability principle needs to be applied during the Service-Oriented analysis and design stages and should be done for all services, especially the agnostic and entity services. Application of this principle at a later stage will most probably negatively affect the service quality because, with delayed documentation, the subtle interpretability details are bound to be lost. Those involved with the early design of the service are most suited for providing this documentation while it is still fresh in their heads (Mukhtar, 12/2011).

2.7.2. Goals

A meaningful application of this principle allows for (Mukhtar, 12/2011):

a. Increase discoverability of services. Enable disparate project team members, with different level of technical expertise, to effectively carryout the discovery process.

b. Increase interpretability of services. The purpose and capabilities are clearly and effectively expressed so that they can be interpreted and understood easily and quickly, both by humans and by programs.

2.7.3. Design-Time vs. Runtime Discoverability

UDDI is the core industry standard behind the service discoverability principle. Whereas there is difference between the idea of a registry and a repository (Figure 24), most modern commercial products package both into a single product for marketing purposes.

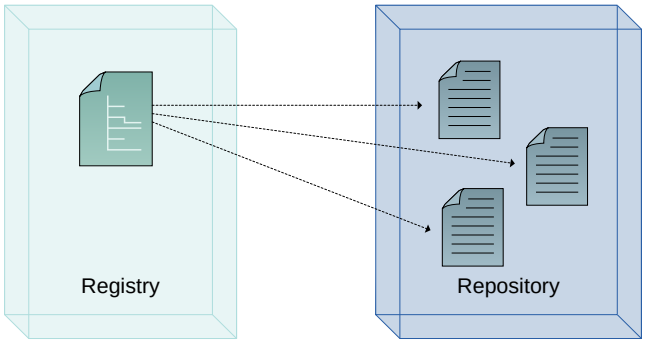


Figure 24: Registry vs. Repository. Registries hold references to architectural artifacts, and repositories hold those artifacts. In other words, the registries hold metadata while the repositories hold the data. Many commercial vendors combine the two ideas into one product. (Mukhtar, 12/2011)

The design-time discovery of reusable services by disparate project architects for potential reuse is most common and an extremely important activity in all successful SOA implementations. However, the runtime dynamic service discovery by intelligent programs hasn't lived up to the promises. The question whether the dynamic service discovery technology is not yet mature in this area, or whether it's simply a false hope, is out of scope for the current concern. What does concern us, however, is the fact that the

mere use of a commercial registry product is not important – the effective dissemination of the relevant information is.

It is well understood and accepted fact that out of the three industry standards that initially made-up the core of the first-generation (or primitive) SOA implementation through the use of web services, namely, WSDL, SOAP and UDDI, the last has lost its prominence (Erl, 2008). Many organizations now prefer to do without a formal COTS registry and repository product because they find little practical value in it. Such organizations instead employ other effective information sharing means, like standardized Service Profiles implemented as a wiki, for the design time discovery of services and sharing of other metadata across disparate project teams, within and outside of their organizations. From the service design principle's perspective, this works just fine. As stated earlier, the goal of this principle is to improve the communications quality of service metadata, and not the use of a certain set of COTS products.

While design-time discovery and interpretation is currently most common, there's work being done to achieve some run-time discovery as well. *WS-MetadataExchange* specification, for example, is a step in that direction. By implementing this specification, a service consumer can request the latest version of the technical service contract at runtime (Guinard, Trifa, Karnouskos, S. S., Spiess, & Savio, 2010.3).

2.7.4. Types of Relevant Metadata for Discoverability

Out of the four metadata types defined under Service Abstraction principle, Functional and Quality of Service (QoS) are naturally relevant for discoverability principle as discussed below (Mukhtar, 12/2011).

2.7.4.1. Functional Metadata

The documentation of a service and its capabilities in the form of labels and annotations explain what is being offered. This information is very helpful for architects and solution designer from across project teams in discovering and understanding service capabilities. The more comprehensive and clearer this metadata, greater would be the chance of its getting discovered and reused across projects. Not only that this documentation should be presented in a standard format that helps people to quickly get to what they are looking for, it should be designed to be consumed by non-technical team members as well. Consider that not all solution designers and architects necessarily come from a development background.

2.7.4.2. Quality of Service (QoS) Metadata

Policy data related to the run-time service behavior makes up the QoS metadata. It can include service policies related to operational threshold that might provide information by peak and off peak hours, comments about over all service robustness and performance expectations or SLAs. These and other similar factors can be very helpful in increasing the service interpretability. With functional metadata alone, someone looking for a specific service capability can find your service but that information perhaps will not be sufficient to make a decision if the service is a suitable fit for their composition. QoS bridges the gap and provides that missing behavioral information that architects need to decide to reuse a service.

The following quick checklist can be used to measure the extent the principle of discoverability has been applied (Mukhtar, 12/2011).

a. Document functional metadata in plain language

b. Append service contract with standard, clear and effective meta information
c. Create and socialize a comprehensive service profile in the form of a wiki, or a central service registry
d. Ensure business centricity of the service context by active involvement of the business experts and stakeholders
e. Document QoS metadata in nontechnical language and incorporate it in formal SLA documents
f. Ensure that all functional and QoS related metadata documented within the service contract and in formal SLAs is also documented in service profile and/or service registry.
g. Ensure all documentation is according to the conventions and standards defined at the domain or the enterprise level

2.7.5. Service Discoverability Principle vs. Service Abstraction Principle

As discussed earlier under the Service Abstraction principle, it is beneficial to hide some information regarding the service from its consumers. Service discoverability principle, however, argues in the other direction; to share information beyond the technical contract. It seems contradictory, and to some extent, the two principles do push in the opposite direction.

A finely calibrated balance is needed between the two valid but opposing principles. On one hand, the detailed information about the capabilities of a service needs to be published and shared widely in order to make it an easy candidate for reuse, on the other hand, information that will encourage the potential consumers to tightly couple with a service implementation needs to be masked. Such information hiding, and the resulting

loose coupling, allows independent evolution of the both the service consumer and the service provider. An architect needs to weigh in the risk and potential benefit from both perspectives, and decide which information to publish and which information to withhold. An improper balancing of these two principles at the design time will not be revealed until long after the service has been active in the production environment. Thus, it might be quite late before someone finds the real result of this design decision (Mukhtar, 12/2011).

2.8. Service Composability

2.8.1. Historical Underpinning in Conventional Software Design

SOA is a descendant of the logical evolution of the software modularization techniques that go back more than 50 years. A service's composability is related to its modular structure. Modular structure enables services to be assembled into applications the developer had no notion of when designing the service. (Valipour, Amirzafari, Maleki, & Daneshpour, 2009). In one of his seminal papers, Booch states that object-oriented development is an approach to software design in which the decomposition of a system is based upon the concept of an object which mirrors our model of reality, while the functional decomposition is achieved through a transformation of the problem space (Booch, 1986). SOA is profoundly influenced from that concept, albeit spins the focus from the "model of reality" to the "business utility" in decomposing the problem space. However, a key challenge that SOA architects face is to determine the most appropriate level of service granularity depending upon the granularity attributes e.g. reusability, composability, complexity and business value (Bu, 2011).

2.8.2. Definition and Discussion

The principle of Service Composability may be defined as “Services are effective composition participants regardless of the size and complexity of the composition” (Erl, 2008, p. 392).

As Blum et al summarizes, SOA represents a model in which functionality is decomposed into distinct units (services), which can be distributed over a network, and can be merged and orchestrated together as an application to fulfill some business requirement. These services communicate among each other by passing data from one service to another, or by coordinating an activity between two or more services (Blum, Magedanz, Schreiner, & Wahle, 2009; Garcia-Valls, Perez-Palacin, & Mirandola, 27-30 Jul 2014). The primary concern of the principle of Service Composability is to build services in such a way that they become effective and efficient composition members, irrespective of whether they are immediately required to be part of a composition or not, thus making “the whole greater than the sum of its parts” (Chang, Mazzoleni, Mihaila, & Cohn, 2008). This potential ability of a service is critical for the whole Service-Oriented endeavor because it enables the Service Reusability principle (Papazoglou & van den Heuvel, 2007). Aggregating smaller capabilities from disparate sources to solve a larger problem lies at the heart of the distributed computing paradigm. Service-Orientation formalizes the same methodology into a core design principle (Chang, Mazzoleni, Mihaila, & Cohn, 2008).

Although, the principle of service composability is one of the three design principles that does not directly produce a specific characteristic but regulate and support other principles to produce the desired results, it does directly facilitates the principle of

reusability. Also, this principle is unique in the sense that all other principles, directly or indirectly, support and enable the implementation of this principle (Mukhtar, 12/2011).

The principle of service composability advocates the design of services such that they can be composed and recomposed in creative combinations over a long period of the life of a service. This ability "to correctly and efficiently assemble solutions by composing existing services" that are highly optimized and can sustain multiple and simultaneous compositions "is essential to" the overall success of SOA (Chang, Mazzoleni, Mihaila, & Cohn, 2008). For the services to be repeatedly reused and recomposed they must possess a highly effective execution environment that efficiently manages high concurrency. The service contract needs to be less restrictive (coarse-grained validation) to allow similar (but not same) data exchanges for like functions. The core service logic needs to effectively manage any unnecessary state data that would impede its reliability and scalability (Mukhtar, 12/2011).

2.8.3. Composition Actors and Concepts

Below is a list of important actors and concepts that must be understood for a formal discussion and proper implementation of this design principle (Figure 25) (Mukhtar, 12/2011).

2.8.3.1. Service Composition Related Concepts

a. Service Activity
b. Composition
c. Simple Composition
d. Complex Composition

2.8.3.1.1. Service Activity

Service Activity may be defined as the mapping of the runtime path of the message exchanges between service capabilities that are participating in a service composition. For instance, if in a certain service composition, service A has a capability x, service B has a capability y and service C has a capability z, invoked respectively starting with x through z, the Service Activity can be mapped as $A(x) \rightarrow B(y) \rightarrow C(z)$. This means that Service Activity only includes the interaction between services but not the actions and processes inside the service boundary.

2.8.3.1.2. Composition

Service Composition may be defined as an aggregation of two or more service capabilities. These aggregations are mostly associated with the automation business processes that require specific workflow logic. Service compositions can be further categorized as simple or complex as discussed below.

2.8.3.1.3. Simple Composition

As the name suggests, a simple Service Composition is a relatively simple aggregation of only a few service capabilities to automate not a very complex business process. This kind of Composition usually does not require Business Process Modeling (BPM) nor does it require special business process orchestration environments that include, for example, Distributed Transactions. Most organizations, in the beginning of their SOA adoption phase, start with such simple Service Compositions which in time usually grow into Complex Service Compositions as discussed below.

2.8.3.1.4. Complex Composition

A Complex Service Composition is a relatively advanced aggregation of service capabilities that automates an elaborate business process including such advanced steps as conditional branching and comprehensive exception handling, use of context and transactional management systems and extensive use of SOAP headers. The use of these advanced features often requires Business Process Execution Language (BPEL) and a special execution environment.

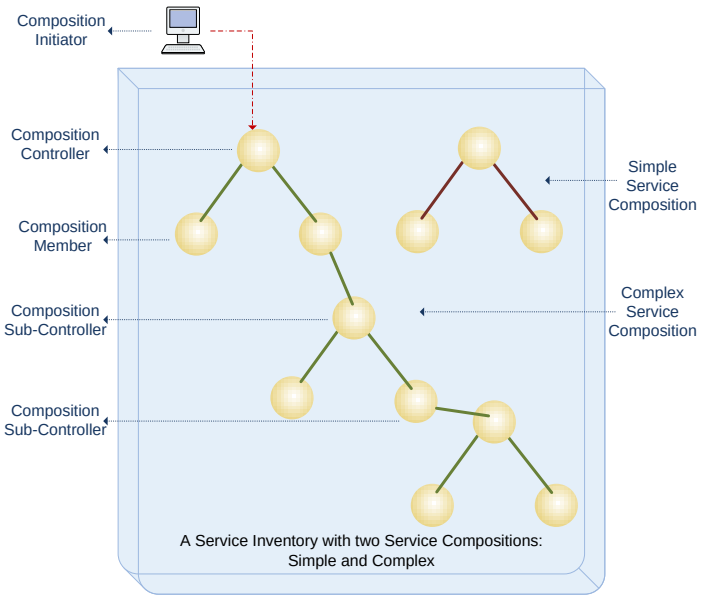


Figure 25: The theory of *Separation of Concerns* advocates decomposing bigger problems into smaller manageable chunks (concerns) without overlapping functional boundaries. The ultimate target-state characteristic of Service-Orientation is a Service Inventory where a large number of services are agnostic, and can be used and reused in creative combinations of complex Service Compositions. (Mukhtar, 12/2011)

2.8.3.2. Service Activity Related Roles

a. Composition Initiator

b. Composition Member

c. Composition Controller

d. Composition Sub-Controller

2.8.3.2.1. Composition Initiator

Composition Initiator is basically the trigger that fires off a service composition at runtime. It's not part of the composition but an outsider that simply passes the required parameters and starts the execution by calling the Composition Controller (as discussed below). Some examples of a Composition Initiator could include a user executing a command from a mainframe command-line, a scheduled desk-top program, a batch program or browser based user sending a request through a web server.

2.8.3.2.2. Composition Member

Composition Member is the service that contains the capability that is being composed, either by a Composition Controller or by the Composition Sub-Controller. In a complex service composition there are often many members working together to complete a predefined business process.

2.8.3.2.3. Composition Controller

A Composition Controller is the service that is at the top of a composition hierarchy. This top level service contains the capability that receives its execution command and parameters from the Composition Initiator (as discussed above) and composes other Compositions members, even possibly Composition Sub-Controller(s) as discussed below.

2.8.3.2.4. Composition Sub-Controller

A Composition Sub-Controller is the service that contains a capability that is composing other service capabilities, but at the same time, is also being composed by

another service. This leads to possible hierarchies of service compositions which are not unusual in well-developed and mature SOA environments.

3. Summary of the Literature Review

The previous two sections examined the research and industry literature for the SOA maturity models, and the principles of SOA design paradigm. Although, some of the prominent existing SOA maturity models are reviewed, it is by no means a comprehensive list of such prevailing models. The nine models examined are only a fair representation of the trend. In the SOA principles section, the eight principles are discussed in depth upon which the industry and research community has converged over time.

4. Related Works

In 2009, Gerić & Vrcek presented a paper in which they outlined a comparative analysis of some existing service-oriented architecture maturity models (SOAMMs). The goal of their study was to find out if different SOAMM's, and their maturity levels are compatible and equivalent; how do different SOAMM correlate, and do they define similar or the same levels of SOA implementation? Their analysis shows that the SOAMMs they compared define very similar maturity levels, and a very similar set of prerequisites that an organization has to achieve in order to increase its maturity level of SOA implementation. They conclude that it is possible to define a basic set of criteria, as a necessary set of prerequisites that an organization has to establish if it wants to establish successful SOA implementation. The question of exactly which set of criteria should be used by an organization was said to depend on the specific domains, i.e. public

administration, manufacturing, retail, financial institutions, etc., and was left open for further research (Gerić & Vrcek, 22-25 Jun 2009).

In section 1 of this chapter, nine different existing SOA Maturity Models from the industry and the research literature where examined. This is not an exhaustive list by any means. Mazzarolo at el, for instance, present several other maturity models in their recent paper. These same researchers also performed a similar study in a large institution in periodic evaluations (Y 2011, 2012 and 2014). The application of their proposed model allowed evaluating the evolution of the level of maturity in each adoption cycle, but was primarily based on the maturity scale of CMMI (Mazzarolo, Martins, Toffanello, & Puttini, Jan. 2015). The current study, on the other hand, is concerned, in the first instance, about determining the correct understanding of the SOA design paradigm within a large federal government agency, and then devising a measuring tool which could also inherently act as an educational tool for this agency.

Architectural principles of component technology are so fundamental to software construction in general that they can now be found across numerous application domains, from traditional desktop applications to enterprise and embedded systems. Buchgeher and Weinreich present a toolkit supporting the design, analysis and implementation of component-based software systems which could be helpful in the design, analysis and implementation of fine-grained services (Buchgeher & Weinreich, 2009).

In 2010, Gu & Zhang offered an SOA based Enterprise Application Integration (EAI) approach in which they offered types of services based on service granularity instead of service concerns (Gu & Zhang, 29-31 July 2010). However, modeling services simply based on their granularity could be counterintuitive, and could limit service reuse. A

modeling scheme based on the theory of separation of concerns, as offered by Erl and others, as explained in Chapter 1, might be preferable.

A study conducted in 2011 by Rostampour at el confirmed that to deliver business agility with SOA effectively, business services should be designed according to SOA principles that affect business agility, including autonomy, cohesion and structural complexity. In order to guarantee service effectiveness towards business agility, this paper offers a set of metrics to evaluate services at the modeling level which are provided from structural complexity, autonomy and cohesion point of views. The researchers analyzed the role of the selected design principles in improving the business agility part of the SOA goals that are outlined in Chapter 1 above (Rostampour, Kazemi, Zamiri, Haghighi, & Shams, 2011).

Chapter 3: Research Methodology

1. Significance of the Research Questions

The four pointed questions raised at the end of Chapter 1 are significant to a general researcher in the area of software design, but are doubly important for an organization that is struggling on the path of SOA adoption. Having made substantial financial and resource investments on the road of SOA adoption while unable to produce matching business results to justify continued pursuit, these struggling organizations are looking for answers.

The research questions are as following.

a. Is SOA largely misunderstood at an organization that struggles in reaching a reasonable level of SOA adoption maturity, and in producing comparable business results?
b. Is the lack of measure of SOA adoption maturity at the Service Architecture level a major cause of the perceived failure of SOA design paradigm?
c. How to measure the level of SOA adoption maturity at the Service Architecture level?
d. How can this more pointed maturity-measure of SOA adoption actually help an organization progress to a higher maturity level?

2. Research Bed Selection

In order to explore these questions, a combination of quantitative and qualitative research methodology is used in this study, targeted to an organization with the following specific prerequisite characteristics of interest:

a. Size of the enterprise must be large enough to rule out individual (or small group) biases. A very large organization (5000 or more employees) would be preferable.
b. The enterprise must have a substantially large budget for its Information Technology organization (preferably in millions of US$) in order to avoid unreasonable financial constraints on new and innovative technology adoption.
c. The enterprise must have a formal and mature management structure with formal structured funding mechanism.
d. The IT organization must have reasonably advanced technical competency, preferably with specialized organizational roles of Enterprise Architects and Solution Architects among its software engineering workforce.
e. The IT organization of this target enterprise must have been working on SOA adoption initiative at some level of the organization for a minimum of five years.
f. There exists a discrepancy between the measured (or perceived) SOA maturity and the measured (or perceived) business results derived from its SOA adoption initiative, i.e. higher SOA maturity, but lower business results.
g. There should be some known general dissatisfaction within the organization with the pace, progress and/or results from the SOA adoption initiative.

Based on the above outlined characteristics, a large independent agency of the US Federal Government (henceforth "Agency") is chosen as the research bed for this study. From within the Information Technology organization of this enterprise, two sets of

pertinent individuals (henceforth “Subjects”) are chosen for direct surveys and interviews. The selection criterion for these Subjects is as follows:

2.1. Subject Group 1

a. Must be directly related to the Information Systems organization
b. Must be in a management official (or higher) position with an intimate understanding of the mission and goals of the IT organization
c. Must have been involved (directly or indirectly) in the past with at least one of the following three areas of specialization:
    I. Enterprise Architecture
    II. Solutions Architecture
    III. Applications Architecture
d. Must have had some past experience with the SOA adoption initiative within the organization

2.2. Subject Group 2

a. Must be directly related to the Information Systems organization
b. Must be in a senior developer, senior analyst, engineer, architect or higher technical position with some understanding of the goals of the IT organization
c. Must have been directly involved in the past with at least one of the following three areas of specialization:
    I. Enterprise Architecture
    II. Solutions Architecture
    III. Applications Architecture

d. Must have had some direct experience with the SOA adoption within the organization

Based on the above listed characteristics, a relatively small group of Subjects (10 individuals) is selected from Group 1, and a larger group (25 individuals) is selected from the Group 2 for the purposes of this study.

3. Survey Questionnaires and the Interview

In order to solicit direct feedback from the Subjects involved in the SOA adoption at this Agency, three separate survey questionnaires is used. Each questionnaire is geared towards providing some insight to the four research questions as described below.

3.1. SOA Adoption Priorities Survey

The *SOA Adoption Priorities Survey* is the first survey questionnaire which contains eight brief multiple choice questions – numbered from Q1 through Q8. An average Subject is expected to take no more than 6 minutes to complete all questions. This survey is designed to gauge the general understanding of the SOA design paradigm by probing into the priorities of the Subject for SOA adoption. This survey helps directly answer the first research question, i.e. *Is SOA largely misunderstood at this organization?* The survey is given to both Subject groups.

Table 1: The SOA Adoption Priorities Survey Questionnaire

SOA Adoption Priorities Survey		
In terms of SOA design paradigm, how important is to:		
Q1	Establish formal processes in the organization	___ Low ___ Medium ___ High
Q2	Have an SOA hardware/software infrastructure	___ Low ___ Medium ___ High

Q3	Establish an SOA governance framework	___ Low ___ Medium ___ High
Q4	Have SOA related tools and technologies	___ Low ___ Medium ___ High
Q5	Have one (or more) ESB as part of the enterprise SOA infrastructure	___ Low ___ Medium ___ High
Q6	Work with an SOA product vendor that is well established, stable and reputed in the industry	___ Low ___ Medium ___ High
Q7	Have one (or more) Enterprise Services Registry as part of the enterprise SOA infrastructure	___ Low ___ Medium ___ High
Q8	Individual service design characteristics	___ Low ___ Medium ___ High

3.2. SOA Maturity Survey

SOA Maturity Survey is the second survey questionnaire. It contains eighteen multiple choice questions under five different categories – numbered from C1 through C5. These categories are crafted based on the prior research/knowledge of the Agency, and are based on the existing and popular SOA maturity models of the industry. Answering all the questions in this survey is estimated to take an average Subject no more than 15 minutes. This survey is devised to assess the perceived maturity of the SOA adoption at the Agency by inquiring about the ancillary aspects (i.e. tools, technologies, processes, infrastructure, etc.) while missing the low-level service architecture aspect which is the mainstay of SOA design paradigm. This survey helps shed some light on the second research question, i.e. *Is the lack of measure of SOA maturity at the Service Architecture level a major cause of the perceived failure of SOA?* The survey is given to both Subject groups.

Table 2: SOA Maturity Survey Questionnaire

SOA Maturity Survey		
Please rate the maturity of the following aspects of your SOA adoption initiative as it exits, is acquired and/or adopted in your organization:		
C1	**SOA Management** (includes *Vision*, *Strategy*, *Funding*, *Roadmap*, *Measurement Model*)	
	An SOA *Vision* is formally documented	___ No ___ Somewhat ___ Yes
	An SOA adoption *Strategy* is formally documented	___ No ___ Somewhat ___ Yes
	Funding is available for SOA adoption	___ No ___ Somewhat ___ Yes
	An SOA *Roadmap* is formally documented	___ No ___ Somewhat ___ Yes
	An SOA *Measurement Model* is formally adopted	___ No ___ Somewhat ___ Yes
C2	**SOA Governance** (includes *Roles and Responsibilities*, *Processes*, *Command and Control Structure*)	
	Roles and Responsibilities are well defined	___ No ___ Somewhat ___ Yes
	Processes are well defined	___ No ___ Somewhat ___ Yes
	Command and Control Structure is well established	___ No ___ Somewhat ___ Yes
C3	**SOA Security Architecture** (Includes *Security Architecture*, *Security Infrastructure*)	
	A *Security Architecture* is established	___ No ___ Somewhat ___ Yes
	Security Infrastructure is in place	___ No ___ Somewhat ___ Yes
C4	**Development** (Includes *Change Management*, *ELC Documents and Templates*, *Reference Process*)	
	A formal *Change Management* process is followed	___ No ___ Somewhat ___ Yes
	Enterprise Lifecycle (ELC) Documents and Templates exist and are available	___ No ___ Somewhat ___ Yes
	A formal *Development Reference Process* is adopted by the development teams	___ No ___ Somewhat ___ Yes

C5	**Infrastructure** (Includes *Service Asset Management*, *Service Usage Infrastructure*, *Standardized Development Environment*, *ESB*, *Service Deployment Platform*)	
	A commercial (or Open Source) Enterprise Services Registry product exists for *Service Asset Management*	___ No ___ Somewhat ___ Yes
	A *Service Usage Infrastructure* is acquired to log real-time service utilization data	___ No ___ Somewhat ___ Yes
	A *Standardized Development Environment* is available to, and is used by, the development teams	___ No ___ Somewhat ___ Yes
	One, or more, commercial (or Open Source) *Enterprise Service Bus (ESB)* product is acquired and available	___ No ___ Somewhat ___ Yes
	A *Service Deployment Platform* exists for PROD/TEST/DEV environment for the execution of the services	___ No ___ Somewhat ___ Yes

3.3. Service Architecture Maturity Survey

The *Service Architecture Maturity Survey* is the third survey questionnaire. It contains eight sections following the eight principles of service design – numbered from P1 through P8. Some sections are further decomposed in order to distinctly deal with some fine-grained aspects of that design principle. Each aspect is clearly and concisely explained. Answering all the questions in this survey is estimated to take no more than 20 minutes for an average Subject. This survey is formulated to measure the real maturity of the SOA adoption at the Agency by deep-divining into the micro SOA, i.e. low-level aspects of service architecture which is the core of SOA design paradigm. This survey helps elucidate and explore the third research question, i.e. *How to measure the level of*

SOA adoption maturity at the Service Architecture level? The survey is given to the Subject Group 2 only.

Table 3: Service Architecture Maturity Survey Questionnaire

Service Architecture Maturity Survey								
Please rate the maturity of the *Service Architecture* in your organization against each of the eight principles listed and explained below:								
#	**Principles of Service Design**		**Realization level**					
			Low				**High**	
			1	2	3	4	5	
P1	**Standardized Service Contract** Do your service-contracts follow these two aspects of this principle? Purpose: Achieve a Federated End-Point Layer	**Standardization of Data-Model**: result in contract definitions that share common XML vocabulary defined at the enterprise level. Once these standardized XML schemas define the I/O for each service capability, the need for data-model transformation is naturally reduced, resulting in efficient service activity.						
		Standardization of Functional-Expression: results in naming conventions for the services/capabilities complying with enterprise standard e.g. *Entity* services should be named according to their business entities, and the *Task* services should be named based on the business process the service is automating. Service capability names should include a verb followed by a noun, and that the service capability names should not repeat their service names.						
P2	**Service Loose Coupling** Do your service-contracts impose low consumer coupling, and are themselves decoupled from the surrounding environment?	Positive Coupling – Logic to Contract: Were the service-contracts written before the service-logic?						
		Positive Coupling – Consumer to Contract: Are the service-consumers tightly coupled to the published service contracts?						
		Negative Coupling – Contract to Logic: Are the service-contracts decoupled from the underlying application logic? (hint: if the service-contract was auto-generated						

	Purpose: Achieve flexibility to change, and independent functional-context	using a tool, from an existing underlying implementation logic then your contract is most likely not decoupled)					
		Negative Coupling – Contract to Functional: Are the service-contracts decoupled from the underlying business processes? (hint: if the service-contract was specifically designed in support of a pre-existing business process or an existing consumer then your contract is most likely not decoupled)					
		Negative Coupling – Contract to Implementation: Are the service-contracts decoupled from the underlying implementations? (hint: if the service-contract is tied to implementation specific elements like legacy APIs, vendor specific database functions, physical server environments, network specific paths, file names and user account information then your contract is not decoupled)					
		Negative Coupling – Contract to Technology: Are the service-contracts decoupled from the underlying technology? (hint: if the service-contract itself is not technology agnostic, but instead tied to the implementation technology like Java or .NET then your contract is not decoupled)					
		Negative Coupling – Consumer to Service Logic: Are the service-implementations inaccessible to consumers except via the published service contracts? (hint: if a consumer can simply bypass the service-contract and can connect directly to the core-service-logic, or to the underlying resources like a database, then you're not decoupled)					
P3	**Service Abstraction** Do your service-contracts only contain essential information, and information about your	Deliberately hide service-metadata such that only necessary information is available to the service consumers, and that too, only via published service contracts. All other non-essential information about the internal logic of your service and its capabilities should be abstracted away (hidden) from the consumers					

	services (outside your team) is limited to what is published in the service contracts? Purpose: Enhanced service reusability, service composability and service discoverability	and consumer designers.					
P4	**Service Reusability** Do most of your services expose agnostic logic that can be positioned as reusable enterprise assets? Purpose: increase ROI, increase organizational agility and reduce IT burden	Consider the potential of a service to be reused by the consumers beyond the original requirement for which it is being designed and developed. This is typically achieved via appropriate Business Process Decomposition, and by following a proper Service Modeling scheme.					
P5	**Service Autonomy** Do your services exercise a high level of control over their underlying runtime execution environment? Purpose: increase reliability, consistency, and behavioral	Services should have maximum amount of control possible over their underlying resources and environment. Carefully consider all direct and indirect dependencies that the service will form, and the potential performance impact of such dependencies.					

	predictability						
P6	**Service Statelessness** Do your services minimize resource consumption by deferring the management of state information when necessary? Purpose: increase scalability, availability and performance	Holding and managing excessive context related state data in working memory negatively impacts the service performance. A service should therefore be designed to hold only the necessary state data that it's currently working upon. All other state data should be tucked in from somewhere it can be retrieved quickly and efficiently when needed while the other processing continues.					
P7	**Service Discoverability** Are your services supplemented with communicative metadata by which they can be effectively discovered and interpreted? Purpose: increase reuse	For the service reusability to work, it is imperative that the services be easily locatable and understandable in terms of their capabilities as well as the data-structures exchanged.					
P8	**Service Composability** Are your services effective composition participants regardless of the size and complexity of the	Services should be designed in such a way that they becomes effective and efficient composition members, irrespective of whether they are immediately required to be part of a composition or not.					

	composition? Purpose: increase reuse						

3.4. Interview

After the Subjects complete the *Service Architecture Maturity Survey* they are briefly interviewed to gather their thoughts on the fourth research question, i.e. *How can this more pointed maturity-measure of SOA adoption actually help an organization progress to a higher maturity stage?* The standardized semi-structured interview contains five questions; each supplemented with an open-ended "how" to stimulate free expression of expert thought. Conducting of this interview is estimated to take no more than 20 minutes with an average Subject. This interview is intended to collect subjective opinions for the qualitative analysis and assessment.

Table 4: SOA Adoption Interview

SOA Adoption Interview	
How can this more pointed maturity-measure of SOA adoption actually help your organization progress to a higher maturity stage?	
1	Can this survey help increase the overall understanding of the SOA design paradigm? How?
2	Can this survey be used as a tool to shift organizational focus to the low-level service architecture? How?
3	Can this more pointed maturity-measure help in your strategic planning, including a revised SOA Roadmap, and in developing a more suitable Maturity Model for your SOA adoption initiative? How?
4	Can this measure help in your financial and resource planning? How?
5	Can this tool be helpful in furthering SOA adoption in your organization? How?

4. Data Analysis

Using the quantitative and qualitative methodology, the data analysis will be performed as following.

4.1. Quantitative Analysis

The raw data captured anonymously with the first three Survey Forms (shown in section 3 above) is aggregated and tabulated (in percentages) showing the score that each survey question received. Based on this data, bar-graphs and/or pie charts are drawn in order to highlight the comparative values of each enumeration separately for each of the three survey forms. The graphical presentation focuses on how the data addresses the specific research questions, and does not necessarily cover all of the aspects of the tabulated data.

4.2. Qualitative Analysis

In empirical software engineering studies, a commonly used strategy for combining qualitative and quantitative methods is to extract values for quantitative variables from qualitative data, often collected from interviews, in order to perform some type of quantitative or statistical analysis. This process is called *coding* (Seaman, 1999).

In this study, the *SOA Adoption Interview Form* is used as the *interview guide*, and *field notes* are taken during the interview process by hand (Taylor & Bogdan, 1984). Employing the process of coding, the answers captured, and any new ideas generated with the *SOA Adoption Interview Form* are summarized, synthesized and grouped in the end. The coding is verified by a review of the field notes by an independent analyst. This section does not limit the discussion to the research questions only, but rather broadens

the scope in order to capture free flow of opinions and reflections from the experts of SOA at this Agency.

Chapter 4: Results

This chapter provides the results of this case study conducted in the form of three survey questionnaires and an interview conducted at this Agency. The tabulation of the results data is followed by a quantitative analysis of the significant findings, which in turn is followed by a qualitative analysis from the SOA Adoption Interview.

1. Survey Results Data

SOA Adoption Priorities Survey is the first survey questionnaire. The results gathered are tabulated below.

Table 5: SOA Adoption Priorities Survey – Combined Results

SOA Adoption Priorities Survey - Results - Combined (Group 1 + Group 2)				
In terms of SOA design paradigm, how important is to:				
		Low	**Medium**	**High**
Q1	Establish formal processes in the organization	9%	20%	71%
Q2	Have an SOA hardware/software infrastructure	11%	31%	58%
Q3	Establish an SOA governance framework	9%	23%	68%
Q4	Have SOA related tools and technologies	17%	37%	46%
Q5	Have one (or more) ESB as part of the enterprise SOA infrastructure	9%	26%	65%
Q6	Work with an SOA product vendor that is well established, stable and reputed in the industry	9%	40%	51%
Q7	Have one (or more) Enterprise Services Registry as part of the enterprise SOA infrastructure	11%	43%	46%
Q8	Individual service design characteristics	26%	40%	34%

Table 6: SOA Adoption Priorities Survey – Subject Group 1 Results

SOA Adoption Priorities Survey - Results - Subject Group 1				
In terms of SOA design paradigm, how important is to:				
		Low	**Medium**	**High**
Q1	Establish formal processes in the organization		20%	80%
Q2	Have an SOA hardware/software infrastructure		20%	80%
Q3	Establish an SOA governance framework		30%	70%
Q4	Have SOA related tools and technologies	20%	20%	60%
Q5	Have one (or more) ESB as part of the enterprise SOA infrastructure		30%	70%
Q6	Work with an SOA product vendor that is well established, stable and reputed in the industry		20%	80%
Q7	Have one (or more) Enterprise Services Registry as part of the enterprise SOA infrastructure	20%	20%	60%
Q8	Individual service design characteristics	30%	40%	30%

Table 7: SOA Adoption Priorities Survey – Subject Group 2 Results

SOA Adoption Priorities Survey - Results - Subject Group 2				
In terms of SOA design paradigm, how important is to:				
		Low	**Medium**	**High**
Q1	Establish formal processes in the organization	12%	20%	68%
Q2	Have an SOA hardware/software infrastructure	16%	36%	48%
Q3	Establish an SOA governance framework	12%	20%	68%
Q4	Have SOA related tools and technologies	16%	44%	40%
Q5	Have one (or more) ESB as part of the enterprise SOA infrastructure	12%	24%	64%
Q6	Work with an SOA product vendor that is well established, stable and reputed in the industry	12%	48%	40%
Q7	Have one (or more) Enterprise Services Registry as part of the enterprise SOA infrastructure	8%	52%	40%
Q8	Individual service design characteristics	24%	40%	36%

SOA Maturity Survey is the second survey questionnaire. The results gathered are tabulated below.

Table 8: SOA Maturity Survey – Combined Results

SOA Maturity Survey - Results - Combined (Group 1 + Group 2)				
Please rate the maturity of the following aspects of your SOA adoption initiative as it exists, is acquired and/or adopted in your organization:				
C1	**SOA Management**			
	(includes *Vision*, *Strategy*, *Funding*, *Roadmap*, *Measurement Model*)			
		No	**Somewhat**	**Yes**
	An SOA *Vision* is formally documented	9%	31%	60%
	An SOA adoption Strategy is formally documented	3%	14%	83%
	Funding is available for SOA adoption	26%	23%	51%
	An SOA *Roadmap* is formally documented	11%	37%	52%
	An SOA *Measurement Model* is formally adopted	20%	17%	63%
C2	**SOA Governance**			
	(includes *Roles and Responsibilities*, *Processes*, *Command and Control Structure*)			
		No	**Somewhat**	**Yes**
	Roles and Responsibilities are well defined	37%	43%	20%
	Processes are well defined	37%	43%	20%
	Command and Control Structure is well established	46%	37%	17%
C3	**SOA Security Architecture**			
	(Includes *Security Architecture*, *Security Infrastructure*)			
		No	**Somewhat**	**Yes**
	A *Security Architecture* is established	17%	63%	20%
	Security Infrastructure is in place	43%	31%	26%
C4	**Development**			
	(Includes *Change Management*, *ELC Documents and Templates*, *Reference Process*)			
		No	**Somewhat**	**Yes**
	A formal Change Management process is followed	6%	23%	71%
	Enterprise Lifecycle (ELC) Documents and Templates exist and are available		17%	83%
	A formal Development Reference Process is adopted by the development teams	6%	31%	63%
C5	**Infrastructure**			
	(Includes *Service Asset Management*, *Service Usage Infrastructure*, *Standardized Development Environment*, *ESB*, *Service Deployment Platform*)			
		No	**Somewhat**	**Yes**
	A commercial (or Open Source) Enterprise Services Registry product exists for Service Asset Management		9%	91%
	A Service Usage Infrastructure is acquired to log real-time service utilization data	9%	40%	51%

	A Standardized Development Environment is available to, and is used by, the development teams		11%	89%
	One, or more, commercial (or Open Source) Enterprise Service Bus (ESB) product is acquired and available		3%	97%
	A Service Deployment Platform exists for PROD/TEST/DEV environment for the execution of the services	3%	9%	88%

Table 9: SOA Maturity Survey – Subject Group 1 Results

SOA Maturity Survey - Results – Group 1				
Please rate the maturity of the following aspects of your SOA adoption initiative as it exists, is acquired and/or adopted in your organization:				
C1	**SOA Management** (includes *Vision*, *Strategy*, *Funding*, *Roadmap*, *Measurement Model*)			
		No	**Somewhat**	**Yes**
	An SOA *Vision* is formally documented		20%	80%
	An SOA adoption Strategy is formally documented		10%	90%
	Funding is available for SOA adoption	40%	10%	50%
	An SOA *Roadmap* is formally documented	20%	20%	60%
	An SOA *Measurement Model* is formally adopted	30%	10%	60%
C2	**SOA Governance** (includes *Roles and Responsibilities*, *Processes*, *Command and Control Structure*)			
		No	**Somewhat**	**Yes**
	Roles and Responsibilities are well defined	30%	40%	30%
	Processes are well defined	40%	30%	30%
	Command and Control Structure is well established	40%	40%	20%
C3	**SOA Security Architecture** (Includes *Security Architecture*, *Security Infrastructure*)			
		No	**Somewhat**	**Yes**
	A *Security Architecture* is established	20%	70%	10%
	Security Infrastructure is in place	30%	30%	40%
C4	**Development** (Includes *Change Management*, *ELC Documents and Templates*, *Reference Process*)			
		No	**Somewhat**	**Yes**
	A formal Change Management process is followed	10%	20%	70%
	Enterprise Lifecycle (ELC) Documents and Templates exist and are available		20%	80%
	A formal Development Reference Process is adopted by the development teams		30%	70%

C5	**Infrastructure** (Includes *Service Asset Management*, *Service Usage Infrastructure*, *Standardized Development Environment*, *ESB*, *Service Deployment Platform*)			
		No	**Somewhat**	**Yes**
	A commercial (or Open Source) Enterprise Services Registry product exists for Service Asset Management		10%	90%
	A Service Usage Infrastructure is acquired to log real-time service utilization data	10%	40%	50%
	A Standardized Development Environment is available to, and is used by, the development teams		20%	80%
	One, or more, commercial (or Open Source) Enterprise Service Bus (ESB) product is acquired and available			100%
	A Service Deployment Platform exists for PROD/TEST/DEV environment for the execution of the services		10%	90%

Table 10: SOA Maturity Survey - Subject Group 2 Results

SOA Maturity Survey - Results – Group 2				
Please rate the maturity of the following aspects of your SOA adoption initiative as it exists, is acquired and/or adopted in your organization:				
C1	**SOA Management** (includes *Vision*, *Strategy*, *Funding*, *Roadmap*, *Measurement Model*)			
		No	**Somewhat**	**Yes**
	An SOA *Vision* is formally documented	12%	36%	52%
	An SOA adoption Strategy is formally documented	4%	16%	80%
	Funding is available for SOA adoption	20%	28%	52%
	An SOA *Roadmap* is formally documented	8%	44%	48%
	An SOA *Measurement Model* is formally adopted	16%	20%	64%
C2	**SOA Governance** (includes *Roles and Responsibilities*, *Processes*, *Command and Control Structure*)			
		No	**Somewhat**	**Yes**
	Roles and Responsibilities are well defined	40%	44%	16%
	Processes are well defined	36%	48%	16%
	Command and Control Structure is well established	48%	36%	16%
C3	**SOA Security Architecture** (Includes *Security Architecture*, *Security Infrastructure*)			
		No	**Somewhat**	**Yes**
	A *Security Architecture* is established	16%	60%	24%
	Security Infrastructure is in place	48%	32%	20%

C4	**Development** (Includes *Change Management*, *ELC Documents and Templates*, *Reference Process*)			
		No	**Somewhat**	**Yes**
	A formal Change Management process is followed	4%	24%	72%
	Enterprise Lifecycle (ELC) Documents and Templates exist and are available		16%	84%
	A formal Development Reference Process is adopted by the development teams	8%	32%	60%
C5	**Infrastructure** (Includes *Service Asset Management*, *Service Usage Infrastructure*, *Standardized Development Environment*, *ESB*, *Service Deployment Platform*)			
		No	**Somewhat**	**Yes**
	A commercial (or Open Source) Enterprise Services Registry product exists for Service Asset Management		8%	92%
	A Service Usage Infrastructure is acquired to log real-time service utilization data	8%	40%	52%
	A Standardized Development Environment is available to, and is used by, the development teams		8%	92%
	One, or more, commercial (or Open Source) Enterprise Service Bus (ESB) product is acquired and available		4%	96%
	A Service Deployment Platform exists for PROD/TEST/DEV environment for the execution of the services	4%	8%	88%

The *Service Architecture Maturity* (SAM) Survey is the third survey questionnaire. The results gathered are tabulated below.

Table 11: Service Maturity Survey

Service Architecture Maturity Survey					
Please rate the maturity of the *Service Architecture* in your organization against each of the eight principles listed and explained below:					
#	**Principles of Service Design**		**Realization level**		
			Low	Medium	High
P1	**Standardized Service Contract** **Do your service-contracts follow these two aspects of this principle?** Purpose: Achieve a Federated End-Point Layer	**Standardization of Data-Model:** result in contract definitions that share common XML vocabulary defined at the enterprise level. Once these standardized XML schemas define the I/O for each service capability, the need for data-model transformation is naturally reduced, resulting in efficient service activity.	48%	24%	28%
		Standardization of Functional-Expression: results in naming conventions for the services/capabilities complying with enterprise standard e.g. Entity services should be named according to their business entities, and the Task services should be named based on the business process the service is automating. Service capability names should include a verb followed by a noun, and that the service capability names should not repeat their service names.	80%	12%	8%
P2	**Service Loose Coupling** **Do your service- contracts impose low consumer coupling, and are themselves decoupled from the surrounding environment?** Purpose: Achieve flexibility to change, and independent functional-context	Positive Coupling – Logic to Contract: Were the service-contracts written before the service-logic?	84%	12%	4%
		Positive Coupling – Consumer to Contract: Are the service-consumers tightly coupled to the published service contracts?	24%	24%	52%
		Negative Coupling – Contract to Logic: Are the service-contracts decoupled from the underlying application logic? (hint: if the service-contract was auto-generated using a tool, from an existing underlying implementation logic then your contract is most likely not decoupled)	36%	56%	8%
		Negative Coupling – Contract to Functional: Are the service-contracts decoupled from the underlying business processes? (hint: if the service-contract was specifically designed in support of a pre-existing business process or an existing consumer then your contract is most likely not decoupled)	68%	20%	12%

		Negative Coupling – Contract to Implementation: Are the service-contracts decoupled from the underlying implementations? (hint: if the service-contract is tied to implementation specific elements like legacy APIs, vendor specific database functions, physical server environments, network specific paths, file names and user account information then your contract is not decoupled)	20%	72%	8%
		Negative Coupling – Contract to Technology: Are the service-contracts decoupled from the underlying technology? (hint: if the service-contract itself is not technology agnostic, but instead tied to the implementation technology like Java or .NET then your contract is not decoupled)	68%	16%	16%
		Negative Coupling – Consumer to Service Logic: Are the service-implementations inaccessible to consumers except via the published service contracts? (hint: if a consumer can simply bypass the service-contract and can connect directly to the core-service-logic, or to the underlying resources like a database, then you're not decoupled)	84%	12%	4%
P3	**Service Abstraction** **Do your service-contracts only contain essential information, and information about your services (outside your team) is limited to what is published in the service contracts?** Purpose: Enhanced service reusability, service composability and service discoverability	Deliberately hide service-metadata such that only necessary information is available to the service consumers, and that too, only via published service contracts. All other non-essential information about the internal logic of your service and its capabilities should be abstracted away (hidden) from the consumers and consumer designers.	84%	12%	4%
P4	**Service Reusability** **Do most of your services expose agnostic logic that can be positioned as reusable enterprise assets?** Purpose: increase ROI, increase organizational agility and reduce IT burden	Consider the potential of a service to be reused by the consumers beyond the original requirement for which it is being designed and developed. This is typically achieved via appropriate Business Process Decomposition, and by following a proper Service Modeling scheme.	76%	16%	8%

P5	**Service Autonomy** **Do your services exercise a high level of control over their underlying runtime execution environment?** Purpose: increase reliability, consistency, and behavioral predictability	Services should have maximum amount of control possible over their underlying resources and environment. Carefully consider all direct and indirect dependencies that the service will form, and the potential performance impact of such dependencies.	76%	20%	4%
P6	**Service Statelessness** **Do your services minimize resource consumption by deferring the management of state information when necessary?** Purpose: increase scalability, availability and performance	Holding and managing excessive context related state data in working memory negatively impacts the service performance. A service should therefore be designed to hold only the necessary state data that it's currently working upon. All other state data should be tucked in from somewhere it can be retrieved quickly and efficiently when needed while the other processing continues.	68%	24%	8%
P7	**Service Discoverability** **Are your services supplemented with communicative metadata by which they can be effectively <u>discovered</u> and <u>interpreted</u>?** Purpose: increase reuse	For the service reusability to work, it is imperative that the services be easily locatable and understandable in terms of their capabilities as well as the data-structures exchanged	56%	28%	16%
P8	**Service Composability** **Are your services effective composition participants regardless of the size and complexity of the composition?** Purpose: increase reuse	Services should be designed in such a way that they becomes effective and efficient composition members, irrespective of whether they are immediately required to be part of a composition or not.	52%	36%	12%

2. ***<u>Quantitative Analysis</u>***

Some significant findings from the data gathered from the three surveys are respectively examined below.

2.1. SOA Adoption Priorities Survey

The *SOA Adoption Priorities Survey* was designed to determine the focus of the priorities of the Subjects with respect to the SOA design paradigm. Offering eight discrete priority dimensions, with the first seven being ancillary, and the last being essential to the SOA paradigm, this survey underscored the undue focus of the Subject's priorities on the ancillary aspects at this Agency. Below is a graphical representation of an aggregation of the same from the combined (Group 1 + Group 2) survey results of the "Low" rankings.

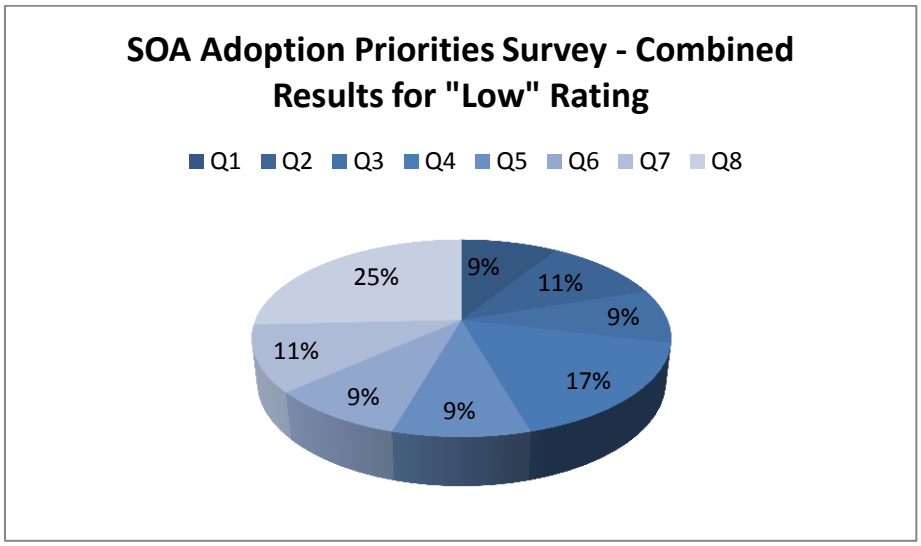


Figure 26: The "Low" Priority Rating Pie Chart – Combined Results (G 1 + G 2)

A substantial number (26%) of all Subjects rated the "Individual Service Design" as Low priority. Also noticeable is the fact that the "High" ranking for the same (34%), although a bit higher that the "Low", comes out to be the least important "High" priority. This result strengthens the hypothesis 1 that SOA could largely be misunderstood at this organization.

2.2. SOA Maturity Survey

The *SOA Maturity Survey* was developed to evaluate the perceived maturity of the SOA adoption at the Agency by inquiring both subject groups about the supplementary aspects of SOA maturity, while omitting the core of SOA design paradigm. This survey helped with the second research question, i.e. *Is the lack of measure of SOA maturity at the Service Architecture level a major cause of the perceived failure of SOA?*

If the Combined Results table is aggregated for each of the five categories in this survey as both Subject Groups marked "yes", noticeable is the fact that except for the Governance and Security categories, the perceived maturity comes out quite high (62%, 72% and 83% for Management, Development, and Infrastructure respectively). This observation becomes even more significant if "somewhat" and "yes" columns are combined (an 86% aggregated score for Management, 60% for Governance, 70% for Security, 96% for Development, and 98% for Infrastructure). These numbers further support the hypothesis that because the SOA maturity at this Agency is not being measured at the Service Architecture level, the perceived SOA maturity is quite high.

Table 12: SOA Maturity Survey - Combined Aggregated Results

SOA Maturity Survey - Results - Combined (G1 + G2) Aggregated				
C1	**SOA Management** (includes *Vision*, *Strategy*, *Funding*, *Roadmap*, *Measurement Model*)			
		No	**Somewhat**	**Yes**
	Aggregated	14%	24%	62%
C2	**SOA Governance** (includes *Roles and Responsibilities*, *Processes*, *Command and Control Structure*)			
		No	**Somewhat**	**Yes**
	Aggregated	40%	41%	19%
C3	**SOA Security Architecture** (Includes *Security Architecture*, *Security Infrastructure*)			
		No	**Somewhat**	**Yes**
	Aggregated	30%	47%	23%
C4	**Development** (Includes *Change Management*, *ELC Documents and Templates*, *Reference Process*)			
		No	**Somewhat**	**Yes**
	Aggregated	4%	24%	72%
C5	**Infrastructure** (Includes *Service Asset Management*, *Service Usage Infrastructure*, *Standardized Development Environment*, *ESB*, *Service Deployment Platform*)			
		No	**Somewhat**	**Yes**
	Aggregated	2%	15%	83%

2.3. Service Architecture Maturity (SAM) Survey

The *Service Architecture Maturity Survey* was devised to measure the real maturity of the SOA adoption at this Agency by deep-divining into the principles of SOA. This survey helped in exploring the third research question, i.e. *How to measure the level of SOA adoption maturity at the Service Architecture level?* Only the Subject Group 2 was

asked to evaluate how closely they followed the principles of SOA while designing their individual services.

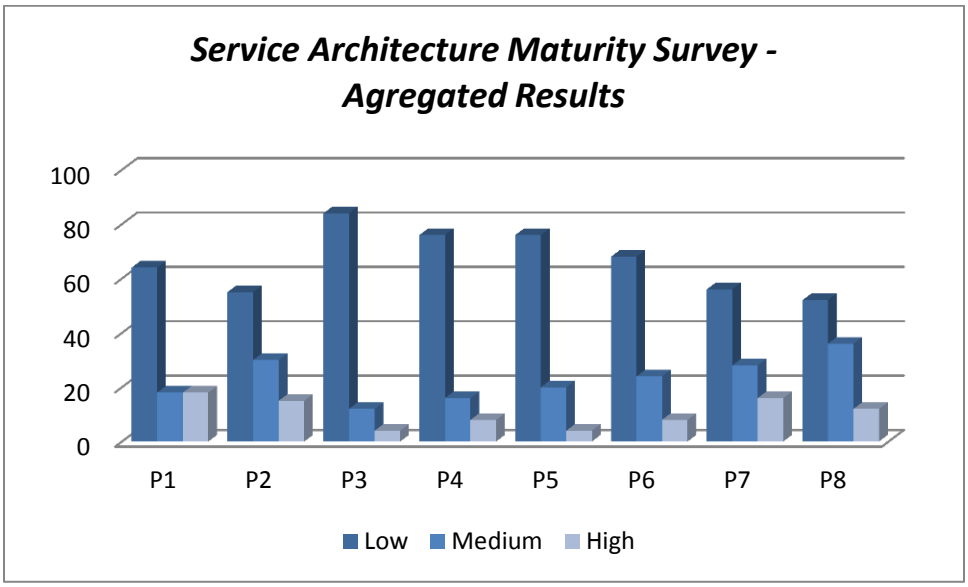


Figure 27: Service Architecture Maturity Ratings for the Eight SOA Principles – Bar Graph

The data from this survey strongly suggest that the principles of SOA have not been given the due consideration while designing the individual services at this Agency. When the Subject Group 2 is specifically asked to evaluate their past solution designs by focusing their attention on the eight principles of SOA one by one, given the precise explanation of each principle, their rating comes out to be mostly on the low end. This finding does not come as a surprise given the results of the survey 1 above, which also suggested misplaced priorities in the overall SOA adoption initiative. Except for the principle of Service Discoverability and Service Composability, all other principles are rated as "Low" implementation by well above 60% of the Subjects. The principle of Service Discoverability and Service Composability fare only marginally better which can potentially be explained by high achievement (83% aggregated) in the Infrastructure

category of the survey 2. This Agency have fairly advanced SOA infrastructure which include a highly rated ESB and a commercial Service Registry which can partially help in implementing these two SOA principles. As can be seen from the bar-graph above, all the eight principles are rated fairly low on the implementation of the past service designs. The "High" rating, on the other hand, is consistently below 10% for all the eight principles except one; i.e. the Standardized Service Contract principle. The aggregated "High" rating received by this one principle stands at 18%; slightly higher but not substantially different from the rest. However, also noticeable is the fact that the "Data Model Standardization" part of this principle individually received a "High" rating of 28%, which is the highest individual rating overall. This achievement can partly be explained by a successful initiative by this Agency of establishing an Enterprise XML Vocabulary group which, to a certain degree, has successfully implemented the Schema Centralization design pattern (Mukhtar, 2011).

3. Qualitative Analysis

Qualitative data are the data represented as words and pictures instead of numbers. The techniques of gathering qualitative data and performing qualitative analysis has achieved significant recognition by the broader software engineering research community in order to study the complexities of human behavior (e.g., motivation, communication, understanding) and solve complex management and organizational issues which are sometimes referred to as "people problems" (Seaman, 1999). The blend of technical and human behavioral aspects in the current case study lends itself to combining qualitative and quantitative methods, in order to take advantage of the strengths of both. Participant observation and interviewing are the two methods used in gathering this type of data.

In this study, a semi-structured interviewing technique is employed to include a mixture of open-ended and specific questions, designed to elicit not only the information foreseen, but also unexpected types of information, mainly to collect opinions and impressions of the Subject Group 2 about the SOA adoption initiative at this Agency. The process of *coding* is used to transforms qualitative data into quantitative data without affecting its objectivity. This coding is further reviewed and verified by an independent analyst who agreed with the results and aggregations.

The findings from the qualitative part of this study are analyzed, synthesized, and aggregated below. The sections used here are simply the most natural and intuitive grouping of the gathered information.

3.1. Disparity in Current Measure of SOA Maturity

After taking the SAM survey, 84% of the Subjects felt that it was helpful in focusing them on the real SOA. A majority of them also stated that they can now see how their focus has been misplaced on technology rather than on the service design principles. While 72% of the Subjects felt that their existing SOA maturity model needs improvement and should be incorporated with SAM, 28% stated that SAM independently stands out as a better model for taking a more pointed measure. 80% of the Subjects agreed that a periodic measure of SOA maturity should be conducted using SAM (or SAM incorporated into their existing measurement model); 32% voted for a yearly measure, while 48% considered a survey every two years to be appropriate. A small minority (16%) felt skeptical that SOA adoption maturity can ever be measured accurately with any model simply due the sheer complexity involved.

3.2. Education and Training

A majority of Subjects were found sensitive and frustrated due to the lack of technical training opportunities in SOA. A large majority (72%) expressed interest and motivation for attending focused SOA training and certification, if provided. However, almost all of these Subjects mentioned lack of funds and ongoing budget constraints as the explanation for almost nonexistent educational opportunities. Some of them jestingly called themselves "paper engineers" referring to the lack of real hands-on training in the field. Several Subjects also mentioned the fact that any on-the-job training that they receive is mostly vendor driven. For instance, their ESB vendor is SoftwareAG which only provides guidance in their own product, i.e. WebMethods. Subjects felt the need to be able to learn about other ESB products available in the market, and get some vendor-neutral training about just the ESBs in general so that they can be better informed and broaden their technological horizon. This observation is pertinently reconfirming the findings of this current study, as the results of the first survey suggested largely misplaced focus in the overall SOA adoption initiative at this organization. If used periodically, every year or every other year, as a maturity measure and as a quality control tool, SAM can fill this educational gap to a certain degree; however, the need for more focused training in the SOA remains valid for this organization.

3.3. Project Based Funding vs. Enterprise Level funding

A strong majority (72%) of the Subjects, in one way or the other mentioned the challenge of project based funding models followed at this Agency. The individual projects bring their own funding and remain tactically focused in meeting their own business requirements. There is little motivation for these projects to follow the SOA

direction and standards set by the enterprise group, especially when following such strategic enterprise direction implies investing more time and resources with little visible and immediate benefit related to their individual project requirements. Some Subjects also mentioned that this particular challenge has been, at least to a certain degree, understood by the senior IT leadership, and thus a new Strategy group at the enterprise level has been stood up recently with its own independent funding stream. However, the larger question of what exactly needs to be funded by this new group and what should remain funded by the individual projects stays somewhat confounding. Although not directly covered under the scope of this current study, this observation is very interesting and significant for the continued future research in this area.

3.4. Too Big and too Complex for SOA

A minority of the Subjects (36%) expressed some level of doubt in the ability of this Agency to ever truly adopt SOA to the fullest, simply due to the inherent high level of complexity involved. They pointed out the large and complex organizational structure and a prevalent culture too inclined to technological inertia to effect such a widespread change as the SOA adoption demands. Some of these Subjects alluded to the absence of will or inability in the senior executive leadership to coordinate an enterprise wide organizational change including the new roles needed for the SOA adoption. They quipped about the decades old roles and technologies which continue to be responsible for running the business critical core systems of their organization to date. This seems a worthy challenge which needs to be looked closely in a future research project extending the current study.

3.5. Strategic vs. Tactical Executive Disposition

Noting that a successful SOA adoption requires strong executive sponsorship, some Subjects (24%) cited the fact of term-appointed senior executives in this Agency. These Subjects felt that such limited time appointment holders (usually for four years) are naturally predisposed to tactical results for proving their achievements, and are not seriously interested in investing in strategic initiatives like SOA which cannot bare substantial fruits within their limited time tenure. Although, no other corroborating evidence was found for such a phenomenon, the challenge is worth looking deeper in an organizational research study, because if it exists, it might not be limited to this one government Agency.

3.6. Ownership Question

The SOA Governance related questions loomed large on the minds of many of the Subjects. One of the questions that kept coming up in the interviews was that of Ownership of the Services. Who owns the Enterprise Common Services[4]? The enterprise group or the individual project teams? Who makes the modification in an existing Service when an additional service capability is needed, not by the owner project team but by some other business domain? Questions like these become more relevant when the funding aspect is brought to bear. The issue really boils down as to who foots the bill for common services, when the funding in this agency has traditionally remained project based. Although, it begs the question, but in a large organization like this one, such governance concerns are very natural. Creating new and independent funding streams for

[4] Enterprise Common Services or ECS is the term used at this Agency for the Services that are delivered under the SOA initiative

organizations like Enterprise Services, and focused training can be recommended, especially at the midlevel management, on the SOA governance track from a vendor-neutral perspective, to mitigate the situation before it gets worse with the further progress in the SOA adoption at this Agency.

Chapter 5: Conclusion

1. Overview

This dissertation began with an introduction to the SOA design paradigm as a unique architectural style which, in the recent past, has gained significant momentum in the information technology industry, and has attracted attention from the research community. Its promised goals and benefits were outlined, and then followed some challenges that organizations face in its meaningful adoption. An unexplained disparity has been observed among some organizations in realizing the promised benefits from SOA adoption – while some were successful, others have not been so fruitful. Around the same time, mostly in the nonscientific literature, a backlash to the SOA popularity was observed, sometimes going so far as declaring that "SOA is dead" (Appendix E). Instead of passing quick judgment and declaring SOA as an empty promise, this study considered it prudent to explore the essence of this architectural style, and separate the necessary from the ancillary aspects.

There are two related areas identified that needed detailed literature review in order to elucidate the problem domain: 1) The principles on which this design paradigm stands and; 2) The existing measurement models of its attained maturity. For the former, eight established design principles have been identified that form the cornerstone of SOA design paradigm. These are the principles that must be kept in consideration when designing individual services. For the later, several popular measurement models, currently established in the industry, have been examined. The key point understood with this review is the fact that none of these maturity models accurately measure SOA

adoption maturity against the SOA principles. Instead these models mostly measured the SOA maturity against the ancillary aspects like technologies and infrastructure.

It is hypothesized that the SOA design paradigm could be largely misunderstood at an organization that struggles in its adoption and stalls in realizing its promised benefits. In order to test this hypothesis, a research test bed, at a large government agency, has been selected according to carefully considered criteria, as outlined in chapter 4. At the same time, it was also decided to measure the perceived SOA adoption maturity within the same test bed. The data resulted from the experiments confirmed the hypothesis.

The first survey results suggest considerable misalignment in the priorities and the individual service design considerations. The second survey results indicate a high perceived maturity of SOA adoption based on an existing maturity model. A Service Architecture Maturity (SAM) survey was then developed which measured the SOA adoption maturity directly against the eight principles of SOA, as discussed in chapter 2. This experiment indicates the real SOA maturity to be substantially lower than the perceived SOA maturity. SAM was further positioned as an educational and guiding tool to help this agency refocus on the necessary aspects of SOA adoption, and thus raise its adoption maturity. For the qualitative part of this research, several engineers, architects and management officials were freely interviewed, soliciting their opinions and perspectives on the effectiveness of SAM and other challenges they face day to day in SOA adoption in general. The findings are synthesized in the Qualitative Analysis section of the Results chapter giving some recommendations on the pertinent observations.

The contributions of this research study are fourfold. It:

I. Diagnoses whether the lack of success is related to a general misunderstanding of the SOA design paradigm
II. Diagnoses whether the perception of failure is due to the use of a unsuitable maturity measure and model
III. Provides a focused and suitable tool for measuring real SOA maturity
IV. Explores the potential of this new tool for helping and furthering SOA adoption

Below, each contribution is discussed in light of respective research question and its relevant findings.

1.1. Diagnoses whether the lack of success is related to a general misunderstanding of the SOA design paradigm

The first research question in the case study can be simplified as: Is SOA largely misunderstood? Exploring this question, the SOA Adoption Priorities Survey was designed and executed with both Subject Groups. Recall, Subject Group 1 mainly comprised of management officials, while the Subject Group 2 included non-managerial technologists.

1.1.1. Findings:

The data from this survey supported the hypothesis 1 that SOA can be misunderstood at this organization. The combined groups survey results in both "Low" and "High" rankings suggest significant inclination (26% and 34% respectively) towards ancillary SOA adoption factors as compared to the given "Service Design" option. Looking at Subject Group 1 and Subject Group 2 separately, the data also show a positive correlation

between the two, implying that SOA adoption is somewhat equally misunderstood at both organizational levels.

1.1.1.1. Correlational Analysis:

The raw data from the SOA Adoption Priorities survey limited to "Low" and "High" ratings is tabulated below which suggest a positive correlation between Subject Group 1 and Subject Group 2, indicating a similar misunderstanding in both subject groups.

Table 13: SOA Adoption Priorities Survey - Correlation Analysis

SOA Adoption Priorities Survey – Raw Data			
Subject Group 1		Subject Group 2	
Low	High	Low	High
0	8	3	17
0	8	4	12
0	7	3	17
2	6	4	10
0	7	3	16
0	8	3	10
2	6	2	10
3	3	6	9
Correlation			
Low		High	
0.52741		0.511277	

1.2. Diagnoses whether the perception of failure is due to the use of a unsuitable maturity measure and model

The second research question in this study is about falsely perceived high SOA maturity at this organization, and can be simplified as: Is the omission of Service Architecture maturity a contributing factor to this false perception? Exploring this

question, the SOA Maturity Survey was designed and executed with both Subject Group 1 and Subject Group 2.

1.2.1. Findings:

The combined data from Group 1 and Group 2 suggest a perception of high SOA adoption maturity in at least 3 out of 5 categories. An interesting aspect to explore would be to compare this false maturity perception among the two subject groups. As noted already, SOA adoption is a strategic initiative with all its goals looking beyond short-term tactical gains. Such an initiative naturally requires strong long-term executive-management commitment and sponsorship. Since the Subject Group 1 of this study consists entirely of management officials, a correlational analysis among the two groups can shed some light on this aspect.

1.2.1.1. Correlational Analysis:

The raw data from the SOA Maturity survey is tabulated below which suggest a positive correlation between Subject Group 1 and Subject Group 2, indicating a similar level of false perception of high SOA adoption maturity exists in both subject groups.

Table 14: SOA Maturity Survey - Correlation Analysis

SOA Maturity Survey – Raw Data					
Subject Group 1			Subject Group 2		
No	Somewhat	Yes	No	Somewhat	Yes
0	2	8	3	9	13
0	1	9	1	4	20
4	1	5	5	7	13
2	2	6	2	11	12
3	1	6	4	5	16
3	4	3	10	11	4
4	3	3	9	12	4
4	4	2	12	9	4
2	7	1	4	15	6
3	3	4	12	8	5
1	2	7	1	6	18
0	2	8	0	4	21
0	3	7	2	8	15
0	1	9	0	2	23
1	4	5	2	10	13
0	2	8	0	2	23
0		10	0	1	24
0	1	9	1	2	22
Correlation					
	No		Somewhat		Yes
	0.831816		0.788575		0.925649

1.3. Provides a focused and suitable tool for measuring real SOA maturity

The third research question in this case study is about measuring the SOA maturity at the Service Architecture level. Subject Group 2 which was more closely associated with the design and architecture tasks on ground was asked to take the Service Architecture Maturity survey. The purpose was to explore how closely were the principles of SOA followed in the service design practice?

1.3.1. Findings:

The data from Service Architecture Maturity survey suggest that the individual services are designed without giving much consideration to the principles of SOA; thereby the discrepancy between the (perceived) high maturity and the (real) low maturity in the SOA adoption could be explained. Based on the Service Architecture maturity, a new, more focused, SOA maturity model can thus be constructed as below.

1.3.1.1. Service Architecture Maturity Model (SAMM)

The aim of this model is to provide a framework that consistently measures an organization's SOA adoption maturity and evolution. This Service Architecture Maturity Model consists of five stages. These stages show the progression of SOA adoption within an organization against the eight principles of SOA. The model is visualized as a two dimensional table. The columns of this table indicate the various SOA Maturity Stages, while the rows list the SOA principles.

Table 14: Service Architecture Maturity Model (SAMM)

	Service Architecture Neutral	Service Architecture Aware	Service Architecture Capable	Business Aligned Architecture	Business Driven Architecture
Standardize Service Contract	No contract standardization effort	Technology standardization only, e.g. WSDL	+ Standardized functional expression	+ Standardized data model	+ Completely standardized contracts, including non-technical part
Service Loose Coupling	Coupling not considered	Minimally considered	Contracts impose low consumer coupling	"Contract First" approach followed	+ Decoupled from the surrounding environment
Service Abstraction	Service architecture openly available	Published contracts	Limited consumer awareness about service architecture	Contracts limited to essential information only	Black-Box services - All service-metadata strictly hidden except what is published in the contract
Service Reusability	Monolithic services	Agnostic vs non-agnostic logic considered	Service modeling scheme followed	+ Logic Centralization pattern applied	+ Contract Centralization pattern applied
Service Autonomy	Direct and indirect dependencies not considered	Performance impact of direct dependencies considered	Performance impact of both direct and indirect dependencies considered	Predictability and reliability increased through the application of design-patters	Maximum control possible over underlying resources and environment
Service Statelessness	Resource consumption not considered	Identified state-data	Performance impact considered	Some state-data management deferred and delegated	Management of all state-data carefully considered against performance impact. Deferred and delegated as needed
Service Discoverability	Service metadata scattered and/or not readily available	Service metadata organized and available	+ Standardized service metadata	Increased awareness of discoverable and interpretable services	Widespread awareness of available enterprise resources with accurate understanding
Service Composability	Non-modular systems	Some decomposition	Decomposed and designed based on a given service modeling scheme	Maximization of agnostic services	Highly optimized services that can sustain multiple and simultaneous compositions

1.4. Explores the potential of this new tool for helping and furthering SOA adoption

The last research question in this case study explores the potential of this proposed maturity model for helping and furthering SOA adoption maturity at this agency. Hands

on practitioners and technologists at this agency were given opportunity to express their thoughts via semi-structured interviews.

1.4.1. Findings

The qualitative data gathered from the interviews suggested that a large majority (84%) of the practitioners felt that SAM is helpful in focusing them on the real SOA, and a similar majority agree that a periodic measure of SOA maturity should be conducted using SAM. This contribution, however, goes beyond just that. It not only provides Service Architecture maturity survey as a tool to periodically measure the real SOA adoption maturity at this organization, it also offers a Service Architecture Maturity Model (SAMM) for the assessed maturity to be plotted against. In a technology roadmap, a model is a necessary ingredient, without which a moving periodic measure cannot be objectively assessed over time. SAMM fills that gap, and provides a more focused and direct model than the ones reviewed in the Chapter 2.

2. Concluding Remarks

SOA is a convoluted design paradigm which, besides the obvious architectural mindset shift, also demands cultural and organizational changes for it to be successful. The standardization aspect alone which it emphases requires a group within the organization to hold a broad and strategic enterprise wide vision. Within the US Government agencies, establishment of an Enterprise Architecture (EA) group is mandated by law, i.e. Clinger–Cohen Act 1996. For the private sector organizations as well, an independent group with such enterprise vision makes perfect sense. When individual projects are exclusively focused on meeting their distinct business

requirements, this EA group can provide necessary guidance and standardization across the enterprise, and can be a primary user and beneficiary of SAM.

There has been some recent talk about *Micro SOA* (Manes, 2013) which in essence is not much different from what is advocated in this study – re-shifting the focus on the Services in SOA. Micro SOA and Enterprise Architecture (EA) have a symbiotic relationship; the strategic impact of SOA and the broad reach of EA strongly suggest a mutually beneficial correlation. Observing the need for Technical Architecture and Enterprise Architecture to coexist and interoperate, Booch recently highlighted the uniqueness of SOA design paradigm, because, as he stated, "architecting a business around the services it provides and architecting a software-intensive system that makes manifest those services are shared goals of the enterprise and the technology" (Booch, 2010, p. 95).

Human beings, at a most fundamental level, are tool makers (Goodall, 1998; Walsh, 1920). When encountered with a challenge, instead of solving it with their claws and teeth, humans tend to invent a tool. Human history at large is a witness to this fact. From the simple club used by the cave dwelling ancestors, to the invention of the wheel which empowered them to perform remarkable tasks, there has always been a tool which propelled human civilization to a new high. These tools tend to become extensions of the human mind; enabling to do things that are otherwise not possible. Although, human history is fraught with such inventions and innovations, some of these tools, like the invention of the printing press around early 15th century really changed the course of history, bringing hitherto unimaginable advancement to human civilization. The

invention of the computer in mid-20th century marks such a turning point, triggering a domino effect, and a revolution which continues to reformat the civilization.

Computing machines are not new in human intellectual history; from ancient abacus to Charles Babbage's Difference Engine in the early 1800s; humans have always been fascinated by the power of numbers, and conscious of their own biological limitations in harnessing it fully. Mankind needed a tool; a tool which, in a way, could become an extension of its neurons; a tool which could remain accurate yet untired and unfatigued when charting our big ideas in massive calculations. The invention of the modern computer filled this gap.

However, from the very beginning of the contemporary Information Systems discipline, the advancement in the hardware domain lead the charge while the progress in the software trailed behind. The hardware platform provided raw jinni like power at the tips of the software programmers to do enormous computations in microseconds. However, the first software architects and engineers intuitively designed their systems much like a human mind works – sequentially. Given a large task, human mind intuitively tries to break it down into smaller more manageable chunks, and execute them as a shopping list – i.e. sequentially, from top to bottom (Miller, 1955). This kind of thinking gave rise to the early programming languages that were Structured and Procedural in nature. In the old software systems, for instance, written in COBOL and C, the program control could be seen flowing mostly from top to bottom, with commands like GOTO and PERFROM tossing the program flow from one part of the program to the other. This kind of architecture survived for a few decades until a paradigm shift occurred.

With the advent of Object Oriented (OO) architecture, software engineers realized that looking at a problem from OO lenses is much more effective and efficient (Booch, 1986). Designing a software solution by following the OO design principles, i.e. Inheritance, Encapsulation and Polymorphism, made their systems much more manageable, flexible and reusable. This was especially true when designing scientific and natural systems since everything in nature is an object. Around the turn of the century, however, the world witnessed the rise of e-commerce. Not only that more and more businesses started harnessing the power of modern computers to build their business systems, but that many business models shifted towards online marketplace. This exposed businesses to the challenges of integration and interoperability which the previously designed closed software systems could not survive. Private businesses were not the only entities impacted by these challenges; government agencies and social media encountered the same questions where the commodity traded was mostly information. These challenges received several responses from the software industry in the form of EDI[5], CORBA[6], etc. but the real paradigm shift started with the advent of SOA. The argument was simple. Just like the premise behind OO architecture was: everything in the nature is an object consisting of attributes and behaviors, so did the SOA argue: everything in the business is a service. Even in the manufacturing industry, different departments and shops could be looked at as providing unique services to the product being produced. Thus, it became reasonable to start looking at the business problems from the lenses of SOA, and architecting software solutions according to the principles of SOA. In a way, OO architectural style evolved into SOA specifically for business use.

[5] Electronic Data Interchange (EDI)
[6] Common Object Request Broker Architecture (CORBA)

Architectural models are curious tools because they provide guidance in the form of constraints and principles but do not provide enforcement mechanisms. In other words, an architect is free to pick and choose from among the principles of an architectural style, and still call it by its original name. The same thing applies in the software architecture. About a decade ago, passing through the initial stages on the Gartner's Hype Cycle, SOA had reached its "slope of enlightenment" by 2009 (Lewis, Morris, Simanta, & Smith, Jan-Feb 2011, p. 58). However, during the same time, it also became an overused buzzword. Software engineers, sometimes not fully conversant with this complex tool, yet eager to adopt it, started following and focusing on parts of SOA, and ignoring the others, all the while calling it SOA (Bloomberg, 2013).

A tool is only as good as the hand that wields it. SOA is a sophisticated tool, designed to solve specific challenges, and to affect particular out comes. When understood and used properly it produces great results, but, on the other hand, when used imprecisely, it results in a waste of resources and causes frustration. Observing such frustration sporadically spread out in the industry, this study hypothesized a potential problem in the hand rather than in the tool, and set out to proffer a more direct and accurate measure of SOA adoption maturity.

The same Gartner analyst that in 2009 cried "SOA is dead" (Appendix E) commented four years later that "Service-oriented architecture has become essential for supporting modern application requirements, including mobile enablement, social integration, data virtualization and cloud computing" (Manes, 2013, p. 1).

3. Future Research

Out of the four surveys conducted at this Agency, the first and the third are generic, and thus, can be used at other federal agencies in order to get a fair assessment of the aspects these surveys attempt to measure. However, the second survey and the final interview need to be tailored to the prevailing circumstances of an organization where they are to be executed. These questionnaires were developed based on the prior knowledge of the environment and the direction of this Agency. With some similar background, these surveys can be customized for use at other organizations as well. Further research is needed for such customization, or even for developing a purely generic set of surveys that could be applied to other agencies.

The initial hypothesis in this dissertation was that SOA is misunderstood, or at least not fully understood, at this organization. Under this hypothesis a measurement tool was built which could simultaneously be used as an educational device for the organization to cure a particular ailment. Under the light of the literature that was reviewed, it was a fairly reasonable hypothesis which was then further reinforced by the results gathered from executing the survey 1, but it is by no means the only possibility for the perceived failure of SOA adoption in another organization. Further research is needed to understand other possible causes of staling SOA adoption maturity if the survey 1 results in an outcome indicating strong and accurate understanding of the SOA design paradigm at another subject organization.

In this study, Subjects were divided into Group 1 and Group 2 based on their affinity of roles as technologists vs. managers, and their closeness to service architecture work during design and engineering stages. The goal was to explore similarities or differences

in their perception and priorities in SOA adoption. A simple correlation analysis is performed for that purpose. However, potential for further exploration remains in order to understand how the two groups within the organizational hierarchy contribute towards SOA adoption. A more fine-grained data collection, followed by some advanced statistical models can shed light on this important aspect in a future research.

Appendices

1. *Appendix A*

This story appeared on Network World. Used here with permission.

Model offers measure for SOA success

By Jon Bachman, special to Network World
February 13, 2006 12:04 AM ET

This vendor-written tech primer has been edited by Network World to eliminate product promotion, but readers should note it will likely favor the submitter's approach.

Network World - Service-oriented architecture has emerged as the most significant shift in how applications are designed, developed and implemented in the last 10 years.

A consortium of software vendors and consultants recently introduced the SOA Maturity Model, which is designed to provide IT decision makers with a framework for benchmarking the strategic value of their SOA implementations and planning. The model is divided into five levels.

Level 1: Initial services
At the initial stage, an organization creates definitions for services and integrates SOA into methodologies for project development. In a financial-services environment, a Level 1 project may use an application server or an enterprise service bus (ESB) adapter to create Simple Object Access Protocol and HTTP Web service invocations between a management system that places an order and a trading service that accepts the order.

Level 2: Architected services
At this stage, standards are set for the technical governance of an SOA implementation, typically under the leadership of the architecture organization. Standard SOA infrastructure and components, such as an ESB, a services and policies repository, an exception-management service, a transformation service and a single sign-on service, are used to foster greater reuse of services, as well as provide tight management and control of services across an organization.

Level 3: Business services and collaborative services
Level 3 features the introduction of business-oriented services, such as business process management (BPM). With a focus on the partnership between technology and business organizations, Level 3 optimizes the flexibility of business processes, allowing IT to respond quickly to changing business requirements. For example, a Level 3 project utilizing BPM might use a Universal Description, Discovery and Integration registry to find a funds-transfer service that could significantly reduce settlement times. This service would be connected to the ESB process within hours of recognizing the business need.

Level 4: Measured business services
Level 4 provides continuous feedback on the performance and business impact of the processes implemented at Level 3. The key focus at this level is collecting data and providing that data to business users, enabling them to transform the way they respond to events.
In our example, a Level 4 project could introduce logging and a service to monitor business activity. These functions provide a collection and display process for business managers to view their trade routing operation and for compliance officers to monitor trading behaviors of their staff and customers.

Level 5: Optimized business services
At this final level, business-optimization rules are added, and the SOA becomes the nervous system for the enterprise. Automatic responses to the measurements and displays of Level 4 allow an organization to take immediate action on events.
A Level 5 project can take the request messages entering the ESB and route that information to an event-stream processor. This service correlates the behavior of all traders across multiple execution venues and identifies important patterns. This information might be used to execute new trades or stop a rogue trader who is out of view of compliance officers.

The SOA Maturity Model provides a framework for IT and business users to properly evaluate the applicability and benefits of SOA in an organization.
Bachman is senior director of product marketing at Sonic Software. He can be reached at jbachman@sonicsoftware.com.

From: Tim Greene [mailto:TGreene@nww.com]
Sent: Friday, September 25, 2015 2:20 PM
To: Mukhtar Gohar
Subject: RE: seeking permission to use an old online article in my PhD dissertation as appendix

Hi, Gohar,
That should be OK.
Best,
Tim

Tim Greene
Senior Editor
Network World
MAXIMIZE YOUR RETURN ON IT
492 Old Connecticut Path
Framingham, MA 01701-9002
508.766.5432
Twitter: @Tim_Greene

From: Mukhtar Gohar
Sent: Wednesday, September 16, 2015 9:51 AM
To: Tim Greene <TGreene@nww.com>
Subject: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Tim,

Hope you are doing well.

I'm seeking permission for using the following full article as an appendix in my Ph.D. dissertation that I'm currently working on at the IS department at UMBC.

> This story appeared on Network World at
> http://www.networkworld.com/news/tech/2006/021306-soa.html

Are you the right person for this? If not, could you please guide me to the right person in your organization?

Thanks,
Gohar Mukhtar

2. *Appendix B*

Gartner

This story appeared on **Gartner Newsroom.** Used here with permission.

Press Release

Egham, UK, May 14, 2009

Gartner Survey Shows 40 Per Cent of SOA Users Don't Measure Time to Achieve Return on Investment

Analysts Explain How to Drive Value from SOA at Gartner SOA & Application Development and Integration Summit 2009, June 24-25 in London

Organisations must set realistic expectations, and identify a few numerical measures of success agreed with the business, to achieve value from service-oriented architecture (SOA), according to Gartner, Inc. A global Gartner survey of 200 companies conducted in the fourth quarter of 2008, found that 40 per cent of SOA users don't measure the time to achieve return on investment, and 50 per cent of non-SOA-users have not adopted it because they cannot articulate and demonstrate the business value of it.

"Many companies come to SOA with excessive expectations, such as immediately achieving quicker project cycles, but users often are not aware of the efforts, resources and time needed to achieve these benefits," said Massimo Pezzini, research vice president and fellow at Gartner. "Consequently, some SOA projects are perceived to have failed when in fact there are simply no well established metrics to evaluate success. Therefore sometimes the benefits are there, but people keep arguing how much better things are, and whether any improvement is really linked to SOA."

"Under the pressure of technology vendors and with a generally too optimistic view of the possible benefits, organisations tend to over-spend on technology but under-spend from an organisational and governance viewpoint, so they come to the conclusion that SOA is expensive and doesn't deliver," said Paolo Malinverno, research vice president at Gartner.

To ensure expectations are realistic and accurate measurement towards goals takes place, Gartner recommends organisations initially focus on achieving just one key benefit from the list of potential business advantages of SOA. As the SOA effort matures, and the benefit starts being delivered, they should add further benefits and change or add to the measures accordingly. Choice of benefit depends on what is the most urgent or important business value that benefit can deliver, and how quickly it can be reached. The benefits include:

Improved Efficiency in Business Processes Execution - Isolating the business logic from the functional application work enables a clearer view of what a process is, and the rules to which it adheres. This can be measured by lower process administrative costs, higher visibility on existing/running business processes, and reduced number of manual, paper-based steps; better service-level effectiveness; quicker implementation of process iterative or of variants of the same process for different contexts.

Quicker Time to Market/Shorter Project Cycles - The SOA principle of modularity results in services than can be reused repeatedly from different contexts. This can be measured by easier internationalisation of processes and quicker adaptation of processes to several products; the number of new products per unit of time; a higher percentage of on-time delivery of products; shorter IT integration of merging entities; a higher market coverage index (the number of countries an organisation sells in, weighted by size of revenue/target market global industry revenue); shortened project delivery times; and number of projects completed per unit of time.

Enablement of New Fast-Growth Business Models – Again, SOA's modular, clearly defined, and

shareable system interfaces, coupled with better application quality delivered faster, enable system scalability that can lead to new business models, such as e-commerce and cloud computing. Typical measures are the number of new processes identified and improved over 12 months; new ways/channels for interacting with customers/suppliers, and traffic/number of transactions associated; shortened time of integration of business partners; number of qualified sales leads; decreasing cost of sales.

Shift in IT Culture From New Developments to Reuse - Services being reused can be developed internally and outside the business (e.g. for B2B document exchange, or to extract CRM data), and used on demand on a SaaS basis. Measures can include SOA's total reuse factor (the number of service consumers/number of services) is increasing; a decreasing number of new service requests to the SOA centre of excellence; a decreasing number of new services added to the repository per month; and an increasing percentage of programmers' incentives based on the number of services they reuse or on the number of effectively reused services they design (not on the amount of software they develop from scratch or maintain).

Lowering Total Cost of Application Development and Maintenance- This cost tends to increase year by year, except when disruptive events (such as a major cost-cutting initiatives) bring it down drastically, causing significant pain within the organisation. A successful and well-governed SOA project brings these costs down, with considerably less organisational disruption. This cost is already measured in every IT department, straight out of IT budget data. Another measure is the reduced number of staff doing application development and maintenance, or the higher productivity of existing staff, such as application development or maintenance.

"Organisations must measure and communicate the success of SOA projects continuously in terms of the positive business outcomes achieved or the negative business outcomes avoided. If no-one knows what SOA is good for, it will be seen as just another fashion wave, and the SOA project will be at risk," said Mr Pezzini.

Mr Pezzini added: "Organisations not yet engaged with SOA should avoid attempts at building a long-term business case for SOA. They should instead justify initial and focused investments in SOA in the context of projects aimed at addressing business needs with short to mid-term paybacks. Incremental investments to extend their SOA infrastructure and to strengthen governance processes should also be cost-justified on a project-by-project basis in terms of tangible business benefits."

From: van der Meulen,Rob [mailto:Rob.vanderMeulen@gartner.com]
Sent: Wednesday, September 16, 2015 10:08 AM
To: Mukhtar Gohar; Pettey,Christy
Subject: RE: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Gohar,

Information on our news room is public domain. This means you may use it in your appendix provided you attribute clearly to Gartner and do not alter the meaning of the content.

Best regards,

Robert van der Meulen
PR Manager Gartner
Direct: + 44 (0) 1784 267 892
Mobile: +44 (0) 7739 312 218
Media Hotline: +44 (0) 1784 26 7 738
Skype: bobvdmeulen
Twitter: @bobvdmeulen

From: Mukhtar Gohar
Sent: 16 September 2015 15:00
To: van der Meulen,Rob; Pettey,Christy
Subject: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Rob or Christy,

Hope you are doing well.

I'm seeking permission for using the following (full or partial) article as an appendix in my Ph.D. dissertation that I'm currently working on at the IS department at UMBC.

> This story appeared on Gartner Newsroom at
> http://www.gartner.com/newsroom/id/978712

Are you the right person for this? If not, could you please guide me to the right person in your organization?

Thanks,
Gohar Mukhtar

3. *Appendix C*

SD Times SOFTWARE DEVELOPMENT

This story first appeared on **SD Times.** Used here with permission.

SOA's dead; long live SOA

Alex Handy
January 15, 2010

When Anne Thomas Manes declared SOA dead last January, the enterprise software world stood up and took notice. Manes, a vice president and research director with Burton Group (now with Gartner), declared that SOA and middleware products had worn out their welcome in corporate America, and that SOA as a marketing term was no longer useful. Now, one year later, Manes is still bearish on SOA technology, but very bullish on applying SOA best practices to the cloud.

Jason Bloomberg, managing partner at consulting firm ZapThink, said that SOA has changed from being a market category for software and middleware companies to a set of best practices, and that this shift became prominent in the market towards the end of 2008. He said that SOA was always about best practices, but that the companies hoping to make money on the term had obscured this fact with lots of advertising dollars.

Today, said Bloomberg, these companies have seen their markets dry up, and only a few such companies remain; most SOA companies were acquired by the likes of IBM and Oracle over the past four years. But even IBM has changed its marketing to emphasize SOA consulting, not SOA products, he said.

"What's really shifted, and this is what Anne was getting at, is a shift away from vendor-driven fake architecture projects where you say you want to do SOA, you buy from Oracle or IBM, they install it and you wonder where the SOA went," said Bloomberg.

"Budgets were tight last year, and that helped organizations resist spending money on software to solve the problem. There is now a focus on true architecture to leverage existing infrastructure."

ZapThink used to offer SOA-specific advice, and it even ranked SOA tools in the company newsletter. Bloomberg said that such work is no longer a part of the ZapThink business model, which is now focused on training and consulting rather than purchasing advice.

From SOA to the cloud

Now that SOA has lost its appeal as a buzzword, many companies are pushing their middleware products into the cloud. "Vendors had to do something to convince people to buy," said Manes. "They basically took everything they had for SOA and repackaged it to say, 'This is cloud stuff.' But you can't do cloud computing very effectively at all if you're not also using service-oriented principles.
"The business guys aren't interested in investing in this abstract architecture concept. But at the same time, you have to do it or your systems are going to remain in this quagmire they've been in."

Adam Vincent, CTO of Layer 7 Technologies, said that the software underpinnings of SOA still exist, but they are no longer being sold as SOA panaceas.

"The tenets of SOA still exist. One of those is the idea of creating composite services out of existing applications," said Vincent.

Layer 7 recently entered the enterprise service bus market, but the company was careful not to attach the

"SOA" label
to its new appliance. Rather, it's marketing it as an integration appliance.

Greg Schott, CEO of MuleSoft, is also in the ESB business. “The SOA market got all the hype around these big-bang approaches where the company says, 'We are now going to deploy a SOA.' Now, people are realizing that's not the way to do it. The way you get service reuse is to start with grassroots,” he said.

Focusing on best practices
Bloomberg agrees that SOA best practices are just as relevant in the cloud. He said that SOA can help to make cloud-based applications more flexible. Many companies are now realizing they have to enforce policies and governance on the many APIs they use across the Internet, and this is a problem that SOA best practices have already addressed. Those best practices vary from firm to firm, but they tend to have major aspects in common. And these commonalities are also applicable to the cloud.

These best practices, of which IBM, Oracle and Sun all have their own versions of, include items such as “Architect someplace in the network where policies and governance can be enforced on incoming and outgoing traffic,” and “Develop loosely coupled services, not tightly integrated one-off services designed only for use in a single application.”

“You can't do true cloud computing without architecture. Leveraging best practices to build loosely coupled abstractions: That's a SOA best practice," said Bloomberg. "You won't be able to succeed in the cloud without SOA best practices.

"SOA takes the whole notion of an API one step further. What SOA brings to the table is a loosely coupled vision for interfaces. You can still call it an API, but it’s still more of a loosely coupled service and extraction interface. This requires a higher level of governance. If the service provider makes an update, it shouldn't break any of the consumers. That becomes a governance challenge.”

A year later, Manes remains confident that SOA as a tool-driven concept is still dead, but she also said that SOA best practices don't need to be updated for the cloud. After seven years of consulting and researching SOA, she said that for clients today, she gives mostly the same advice she has been giving on how to do SOA, because, she said, “that's also how you do cloud.”

From: Dave Rubinstein [mailto:drubinstein@bzmedia.com]
Sent: Wednesday, September 16, 2015 1:08 PM
To: Mukhtar Gohar
Subject: Re: seeking permission to use an old online article in my PhD dissertation as appendix

Hello Mukhtar!
As long as the article is credited to SD Times (it is copyrighted material) we would allow this kind of use.
Thanks for the inquiry.
David

On Wed, Sep 16, 2015 at 12:55 PM, Mukhtar Gohar wrote:
Hi David,

Hope you are doing well.

I’m seeking permission for using the following full article as an appendix in my Ph.D. dissertation that I’m currently working on at the IS department at UMBC.

> This story first appeared on SD Times at
> http://www.sdtimes.com/content/article.aspx?ArticleID=34062

Are you the right person for this? If not, could you please guide me to the right person in your organization?

Thanks,
Gohar Mukhtar

5. *Appendix D*

This story first appeared on **CIO.com.** Used here with permission.

Burton Group: SOA is Dead; Long Live Services

Although the word "SOA" is dead, the requirement for service-oriented architecture is stronger than ever.
By David Linthicum
Tue, January 06, 2009

InfoWorld — Anne Thomas Manes put out an interesting post yesterday, declaring that SOA has failed, but there are a few FWBs (features with benefits), that we should still consider.

SOA met its demise on January 1, 2009, when it was wiped out by the catastrophic impact of the economic recession. SOA is survived by its offspring: mashups, BPM, SaaS, Cloud Computing, and all other architectural approaches that depend on "services."

I'm actually having Anne on the podcast this week to talk about this.

So, what do I think? In short, she may have something there, and I've been hitting on these issues for years now.

Indeed, while SOA is possible, and has a bunch of value, most of those out there tasked to implement SOA were doing so with all the talent of trained monkeys, and just could not get down to the basic issues of architecture, instead focusing way too much on technology and the hype. In short, the efforts were focused in the wrong directions and now there is little to show for it.

But perhaps that's the challenge: The acronym got in the way. People forgot what SOA stands for. They were too wrapped up in silly technology debates (e.g., "what's the best ESB?" or "WS-* vs. REST"), and they missed the important stuff: architecture and services.

I was actually hopeful that somehow, someway SOA would transcend some of the practice issues I saw around EAI. Indeed, most enterprises are dealing with a huge mess, and it's a good idea to begin to cleaning things up. Right? SOA was and is a great approach for doing that, but many charged with making SOA work just could not get out of their own way.

The demise of SOA is tragic for the IT industry. Organizations desperately need to make architectural improvements to their application portfolios. Service-orientation is a prerequisite for rapid integration of data and business processes; it enables situational development models, such as mashups; and it's the foundational architecture for SaaS and cloud computing.

So what went wrong?

First, there are not enough qualified architects to go around, and you'll find that most of the core mistakes were made by people calling themselves "architects," who lack the key skills for moving an enterprise towards SOA. They did not engage consultants or get the training they needed, and ran around in circles for a few years until somebody pulled their budgets.

Second, the big consulting firms drove many SOA projects into the ground by focusing more on tactics and billable hours than results and short- and long-term value.

Third, the vendors focused too much on selling and not enough on the solution. They put forth the notion that SOA is something they have to sell, not something you do.

Finally, the hype was just too much for those charged with SOA to resist. Projects selected the technology first, then the approach and architecture. That's completely backwards.

However, this failure does not diminish the need for SOA. Indeed, most enterprise architectures are a mess and are getting messier as the years go by. Moreover, if you just look at the inefficiencies within the current IT infrastructure you can easily make a case for SOA.

Although the word "SOA" is dead, the requirement for service-oriented architecture is stronger than ever.

Can't disagree with that. Good luck.

From: Mike Shober [mailto:mike.shober@theygsgroup.com]
Sent: Friday, September 25, 2015 3:06 PM
To: Mukhtar Gohar
Subject: RE: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Gohar,

You are permitted to reference the article as detailed below.

Best,
Mike

Mike Shober
Content Sales & Licensing

The YGS Group
3650 West Market Street | York, PA 17404
p: 717.505.9701 x 2229 | d: 717.430.2229 | f: 888.608.0288
mike.shober@theygsgroup.com
www.theygsgroup.com

MARKETING SERVICES | PUBLISHING SOLUTIONS | PRINT OPERATIONS

From: Mukhtar Gohar
Sent: Friday, September 25, 2015 2:14 PM
To: Mike Shober <mike.shober@theygsgroup.com>
Subject: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Mike,

Hope you are doing well.

I'm seeking permission for using the following full article as an appendix in my Ph.D. dissertation that I'm currently working on at the IS department at UMBC (purely academic use; no commercial value).

http://www.cio.com/article/474174/Burton_Group_SOA_is_Dead_Long_Live_Services

Since I'm told that I can't include URLs or internet links in my dissertation, I plan to refer it as below.

This story first appeared on **CIO.com**

Burton Group: SOA is Dead; Long Live Services

Although the word "SOA" is dead, the requirement for service-oriented architecture is stronger than ever.
By David Linthicum
Tue, January 06, 2009

Are you the right person for this? If not, could you please guide me to the right person in your organization?

Thanks,
Gohar Mukhtar

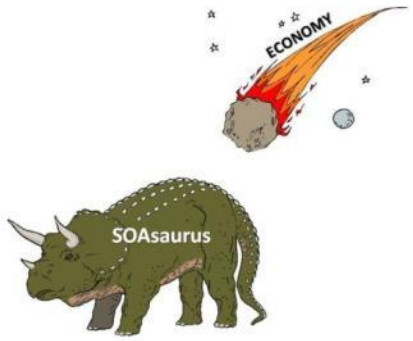
7. *Appendix E*

This story first appeared on **Burton Group** (now **Gartner Inc.**). Used here with permission.

SOA is Dead; Long Live Services

January 05, 2009
Blogger: Anne Thomas Manes
Obituary: SOA

SOA met its demise on January 1, 2009, when it was wiped out by the catastrophic impact of the economic recession. SOA is survived by its offspring: mashups, BPM, SaaS, Cloud Computing, and all other architectural approaches that depend on "services".



Once thought to be the savior of IT, SOA instead turned into a great failed experiment—at least for most organizations. SOA was supposed to reduce costs and increase agility on a massive scale. Except in rare situations, SOA has failed to deliver its promised benefits. After investing millions, IT systems are no better than before. In many organizations, things are worse: costs are higher, projects take longer, and systems are more fragile than ever. The people holding the purse strings have had enough. With the tight budgets of 2009, most organizations have cut funding for their SOA initiatives.

It's time to accept reality. SOA fatigue has turned into SOA disillusionment. Business people no longer believe that SOA will deliver spectacular benefits. "SOA" has become a bad word. It must be removed from our vocabulary.

The demise of SOA is tragic for the IT industry. Organizations desperately need to make architectural improvements to their application portfolios. Service-orientation is a prerequisite for rapid integration of data and business processes; it enables situational development models, such as mashups; and it's the foundational architecture for SaaS and cloud computing. (Imagine shifting aspects of your application portfolio to the cloud without enabling integration between on-premise and off-premise applications.) Although the word "SOA" is dead, the requirement for service-oriented architecture is stronger than ever.

But perhaps that's the challenge: The acronym got in the way. People forgot what SOA stands for. They were too wrapped up in silly technology debates (e.g., "what's the best ESB?" or "WS-* vs. REST"), and they missed the important stuff: architecture and services.

Successful SOA (i.e., application re-architecture) requires disruption to the status quo. SOA is not simply a matter of deploying new technology and building service interfaces to existing applications; it requires redesign of the application portfolio. And it requires a massive shift in the way IT operates. The small select group of organizations that has seen spectacular gains from SOA did so by treating it as an agent of transformation. In each of these success stories, SOA was just one aspect of the transformation effort. And here's the secret to success: SOA needs to be part of something bigger. If it isn't, then you need to ask yourself why you've been doing it.

The latest shiny new technology will not make things better. Incremental integration projects will not lead to significantly reduced costs and increased agility. If you want spectacular gains, then you need to make a spectacular commitment to change. Like Bechtel. It's interesting that the Bechtel story doesn't even use the term "SOA"—it just talks about services.

And that's where we need to concentrate from this point forward: Services.

From: Pettey,Christy [mailto:Christy.Pettey@gartner.com]
Sent: Wednesday, September 23, 2015 4:15 PM
To: Mukhtar Gohar
Subject: RE: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Mukhtar,
You have permission to use http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html as an appendix in your Ph.D dissertation.

Thanks,
Christy

Christy Pettey
Director, Public Relations
Gartner
Tel: 1 408 709 8124
Press Hotline: 1 408 709 8220
E-mail: christy.pettey@gartner.com
Web site: http://www.gartner.com
Gartner Twitter: http://www.twitter.com/gartner_inc
Personal Twitter: http://twitter.com/cpettey
Gartner Newsroom: www.gartner.com/newsroom
YouTube Channel: http://www.youtube.com/gartnervideo

From: Mukhtar Gohar
Sent: Wednesday, September 23, 2015 12:52 PM
To: Pettey,Christy
Subject: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Christy,

Hope you are doing well.

I'm seeking permission for using the following full article as an appendix in my Ph.D. dissertation that I'm currently working on at the IS department at UMBC.

> This story first appeared on Burton Group (now Gartner Inc.) at
> http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html

Are you the right person for this? If not, could you please guide me to the right person in your organization?

Thanks,
Gohar Mukhtar

8. *Appendix F*

This story first appeared on **Information Management.** Used here with permission.

Reality Check: The Costs of Data and Application Integration

By Paige Roberts and David Inbar
NOV 21, 2008 11:24am ET

Disparate data, duplicate data and inaccessible data are all serious problems that can cripple an enterprise. Everyone wants to address them, but budget-conscious corporations can't afford to blow millions of dollars on a project that will disrupt business for long periods and may break within months or weeks of finally being put into place. The sky-high cost of integration projects and timelines measured in months or even years make them the dread of many businesses. If you're unfamiliar with the complexities of integration projects, transferring data back and forth between a few applications, file formats or data stores may sound deceptively straightforward.

Why are Integration Projects so Expensive?
Many factors make integration expensive – some are fairly obvious, some possibly surprising. A Yankee Group report from a few years back, "The Hidden Costs of Data Integration," uncovered some less obvious contributors.1 The report's main conclusion was that while businesses budget for building or purchasing integration tools and setting up initial integration implementations, they rarely factor in long-term maintenance and repair costs. The report notes, "Over a three-year period, the total cost of ownership of an integration application is more than eight times the initial software license investment."

Budget Busting Factors

Staffing costs. According to the Yankee Group, labor is the most expensive aspect of an integration project, costing as much as the software and hardware costs combined over three years' time. This cost as well as the costs associated with disruption of the normal business routine increase as implementation time increases. Anyone who has been involved with integration projects knows that implementation times frequently turn out to be higher than expected. The more those schedules get stretched, the higher the price tag.

One reason for high staffing costs is that the business environment is highly unstable due to mergers and acquisitions, changes in business strategy, new market initiatives and other factors. This instability means that even the most robust integration implementations have to adapt constantly. As much as labor costs incurred during implementation can put a dent in the budget, the costs of making changes can be even more extreme. The periodic downtime of a system that the corporation has come to rely on can cause myriad unpredictable problems that have their own high price tag.

Data quality. Data quality issues are another major cause of system downtime and implementation delays. Even data that everyone involved believes is clean and rigidly regulated frequently has flaws that can cause integration pipelines to crash, driving up downtime and repair costs. If inaccurate or improperly classified data makes it into mission-critical data stores, the result can be faulty analytics, damaged trust and lost business, each of which is costly.

Endpoint variety. One of the biggest contributing factors to integration cost is the wide variety of data to be integrated. Endpoints may include legacy systems on mainframe platforms that have been operating for 20 years or more, while others are software as a service (SaaS) systems that were recently implemented on platforms outside the firewall. In between are a vast array of applications of all types and ages on a variety of platforms with myriad interface types. Because no two programmers solve data storage problems in quite the same way, no two applications ever store their data in the same structure or with the same semantic logic. Even two applications of the same type, like two customer relationship management (CRM)

systems or two accounting systems, can show extreme variations in data storage strategies, making the data incompatible without extensive transformation.

Project complexity. The demands of modern integration initiatives increase the level of complexity of the problem, even while they seek to simplify and improve business processes. Frequently, simplicity for the business user translates to more complexity in the underlying architecture. Data warehouse service-oriented architecture (SOA), business intelligence (BI) and master data management (MDM) projects all propose to make business processes more efficient by connecting multiple disparate systems, but they require complex initial implementations to reach their goals. Meeting the technical demands of real-time updates, bidirectional synchronization, and exponentially increasing data volumes doesn't come cheap.

With multiple factors conspiring to make integration projects into budget busters, it's amazing that anyone accomplishes them at all without ending up in bankruptcy court. Yet, thousands of companies have accomplished very successful integration initiatives, stayed within a reasonable budget and ended up with a much higher ROI than their initial investment. Implementing a solid budget-conscious integration project isn't impossible. You just have to use the right strategies to counter the factors that make integration expensive.

Create an Integration Strategy That Will Not Break the Bank

Adjust your expectations. The first thing that has to happen is that you need to adjust your expectations, particularly of integration tools. Ridiculous price tags for integration projects should not be standard. Bloor Research recently did a survey of approximately 200 companies comparing various integration vendors for price across multiple projects.2 There was a huge difference in initial license costs for integration software, from hundreds of thousands to free for open source tools or those included with other software. Strategies that companies might think would save money are:

1. Invest in more expensive software up front, based on the idea that it would deliver ROI by being more full featured and easier to implement, or
2. Go with open source software or other software that is virtually free and save on up-front licensing costs.

The Bloor study compared various integration tools, including the other factors needed to make those tools work, such as labor, hardware and dependent software requirements. Some of the results were surprising. In particular, open source software, due to high technical resource demands and low reuse potential, was not necessarily the most inexpensive across multiple projects. The high-dollar software was often just as costly long term as it was up front, as much as doubling the cost-per-project relative to other options.

From: Schecter, Peggy [mailto:peggy.schecter@sourcemedia.com]
Sent: Friday, September 25, 2015 5:01 PM
To: Mukhtar Gohar
Subject: RE: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Gohar
Thank you for reaching out
This is approved as long as you credit Information Management Thanks Peggy

-----Original Message-----
From: Gohar Mukhtar
Sent: Friday, September 25, 2015 4:59 PM
To: Schecter, Peggy
Subject: seeking permission to use an old online article in my PhD dissertation as appendix

This message was sent via the form /fdc.collector/Mail Send Form

From_Address: gohar.mukhtar@irs.gov
Contact_Name: Gohar Mukhtar
Subject: seeking permission to use an old online article in my PhD dissertation as appendix

Message: Hi Peggy,

Hope you are doing well.

I'm seeking permission for using the following full article as an appendix in my Ph.D. dissertation that I'm currently working on at the IS department at UMBC.

This story first appeared on Information Management at http://www.information-management.com/infodirect/2008_99/10002234-1.html

Are you the right person for this? If not, could you please guide me to the right person in your organization?

Thanks,
Gohar Mukhtar

10. Appendix G

This story first appeared on **InformationWeek NETWORKComputing.** Used here with permission.

InformationWeek Analytics: State Of SOA

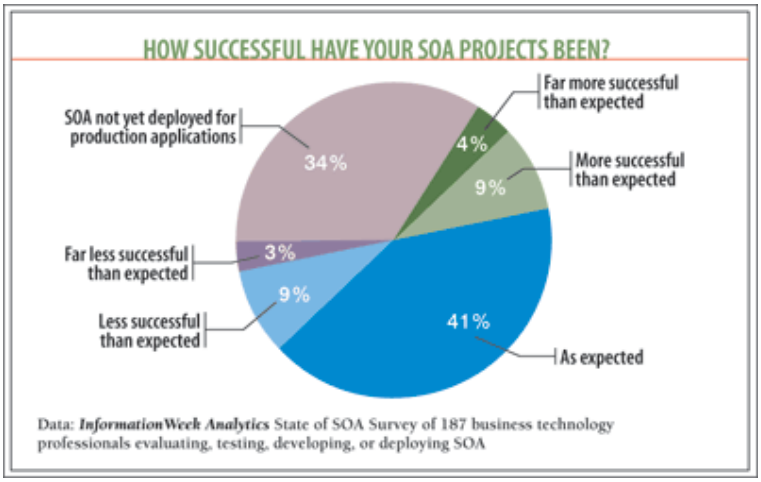
By Roger Smith 02/19/2009

We polled readers to assess whether service-oriented architectures have simply hit an economy-induced bump in the road or are, instead, at a significant crossroads.

Reports of SOA's demise have been greatly exaggerated, according to the 270 business technology professionals InformationWeek Analytics surveyed for this report on the state of service-oriented architecture. But that's not to say there isn't trouble in SOA-ville: Just 23% of respondents say that their organizations have deployed a SOA, and a mere 7% of these report that the resulting systems are available for external use. Twenty-nine percent are experimenting or in development, while 31% have no plans. Much-touted business benefits of SOA, such as increased flexibility and business agility, reduced costs, and improved time to market, weren't major factors speeding increased adoption. The percentage of overall software reuse within organizations rose by just 7 points after initiating a SOA project, from 32% to 39%.

SOA governance, tragically, is DOA.

Still, enterprise IT groups rarely turn on a dime, and they don't lightly abandon technology investments and strategic decisions. When asked if their SOA projects have been successful in delivering a positive business impact, respondents overwhelmingly say results were as expected. Both positive and negative extremes ("more successful" and "less successful") rate nearly identical low scores. One interpretation: It's human nature to resist admitting mistakes, so these IT pros are reluctant to cede defeat. But our take--supported by survey results and discussions with a wide range of stakeholders--is that many companies are moving forward with SOA implementations, though a significant number have decided to shift course and take the path of least resistance. In essence, that means building their SOAs on the Web, using Internet-delivered APIs, and swapping in more agile REST-based Web services as a simpler alternative to heavyweight SOAP-based Web services where appropriate. In fact, when asked to indicate their past, present, and estimated future use of SOAP-based Web services vs. REST-based Web services, respondents show a marked drop-off in use of SOAP, from 54% a year ago to a projected 42% in the next 18 months. The number primarily using or considering REST-based Web services is predicted to grow by a proportional amount, from 14% to 24% over the same time frame.



The REST philosophy has simplicity going for it, and when resources get tight, faster and easier usually wins. However, the two styles can complement each other; it doesn't have to be a case of one or the other. A REST-based approach is a natural for data-oriented applications that focus on simple database look-up scenarios. Many apps fit this model, especially on the Web. Another explanation for the increasing

popularity of REST is the growing number of rapid prototyping tools, such as Ruby on Rails, that can be used to build these types of apps.

REST isn't the best solution for all Web services, of course. Our advice: Don't be married to one method or the other. To simplify your application development process and make it more accessible to more people, first consider REST for straightforward operations. Choose SOAP only when your requirements demand it, as with applications that require complex data retrieval operations or network independence. Here SOAP is the more viable option.

An example of an IT pro taking this balanced approach is Ernest Mueller, whose company has experienced rapid growth of its internal all-SOAP SOA implementation. Mueller manages the Web systems team at National Instruments, a supplier of measurement and automation products for engineers and scientists. Two years ago, as part of a business/technology alignment effort, Meuller and a multidiscipline Web architecture team identified two major areas in which NI needed consistent, reusable systems.

"The first was application delivery on the Web, for which we constructed a reusable Java-based delivery platform for our applications to use," Mueller says. "The second was initially framed as back-end access for our Web site to transact with our ERP systems and other internal data and functionality repositories. After some research, we decided that a full SOA tier was the solution. We had some internal Web services, but wanted the additional functionality an ESB and BPEL engine would give us."

Based on these needs, the team selected Oracle's SOA Suite as its platform. Mueller says that while NI's SOA project has been slow to define standards and governance--a trend in our poll--the company is happy overall with its SOAP implementation internally. However, the team is looking at REST as NI starts to expose services externally, thanks to what Mueller sees as REST's better ease of use.

Not Dead Yet

SOA success stories such as National Instruments' notwithstanding, there's a common industry perception that a critical mass of SOAP-based SOA initiatives have failed to deliver their promised benefits and have run out of steam. In response, a range of pundits have weighed in on SOA's future.

At one extreme of sensationalism, Burton Group's Anne Thomas Manes issued a blog post in January declaring, "SOA Is Dead; Long Live Services," and followed up with an open invitation to a wake. A less-dire November report from Gartner found that a growing number of organizations are delaying their SOA adoption plans, and the number of organizations with no plans to adopt SOA has almost tripled, from 6% in 2007 to 16% in 2008. As discussed earlier, the percentage of companies deferring SOA adoption recorded by our survey was even larger.

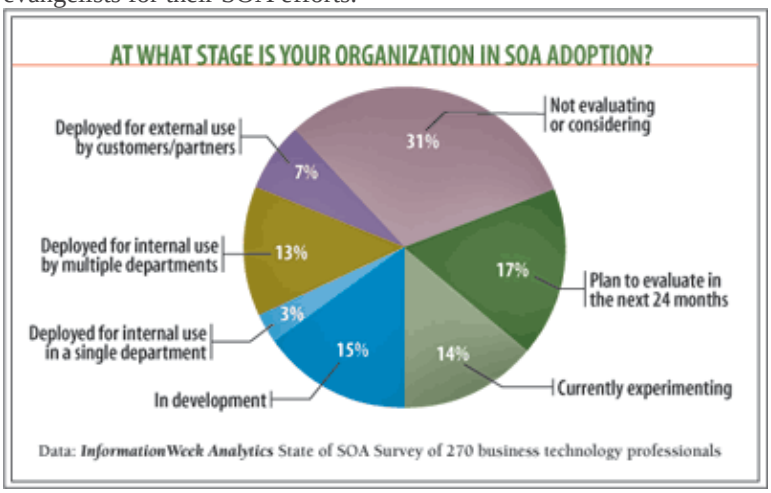
"The biggest challenge is to show to the business the benefits of using SOA," says Krishna Komanduri, a technical director with brokerage firm Charles Schwab. "But, because of the current economic situation, the business isn't enthusiastic about implementing new technologies when it's hard for them to see and realize the benefits. In many cases, [SOA] requires organizational changes both in the business and technology--which is very difficult."

Part of the problem: The percentage of overall software reuse within organizations was only marginally higher after initiating SOA, with a 32% reuse rate cited before the SOA project versus 39% after. The key for maximizing Web service reuse in an enterprise is good SOA governance. However, good governance is hard to find in many IT shops, especially those with outdated incentive structures that encourage developers to write pages of code rather than reuse existing Web services components.

But far and away the major reason respondents who aren't evaluating or implementing SOA cite for not pursuing the initiative is a lack of a viable business case--43% say it's because SOA initiatives have developed a reputation for overpromising and underdelivering.

We're convinced that one of the main reasons for this is that when SOA started out a few years ago, vendors sold the concept to CIOs and other corporate decision makers as being about specific (and expensive) products like Web services or SOA management products, enterprise service buses, SOA gateways, and hardware acceleration devices for Web services. But SOA is about much more than deploying new technology and building service interfaces to existing applications. It requires significant redesign of an enterprise's application portfolio as well as transformation of the entire business. Because so much of SOA is actually about business practices--not technology--in many cases there's been a push back from units reluctant to change or to invest in an IT infrastructure that may require multiple years to pay back the investment.

Other hurdles identified in our survey by nonadopters: that a SOA initiative would increase rather than reduce IT costs and would amplify rather than simplify complexity of the IT environment (17% and 15%, respectively). Roughly the same percentage say they've had difficulty enlisting executive supporters and evangelists for their SOA efforts.



However, the fact that National Instruments is actively considering both SOAP and REST Web services in its SOA implementation is a strong indicator that there's room for these new initiatives and new architectural principles that have value under the broader service orientation umbrella. Whether SOAs are implemented in REST, SOAP, or a combination of both, we believe that a snowball effect will arise over the coming years: As more Web services can be invoked, more applications will be written to invoke them. With the increased availability of Web services components, application designers will evolve from thinking about application architectures as monolithic, siloed software efforts and move toward the exploitation of configurable, component-based SOAs.

NI's Mueller says some of this is happening already; in fact, he became a victim of his own success when the SOA project team was forced to fight for money and resources from other groups. Meuller explains that when his Web architecture team started work on the SOA project, it became clear that the demand at NI for SOA stretched way past the original audience. Immediately, internal groups that were working on projects requiring heavy interoperability came around and wanted in.

"After about six months of work, we went live in June 2008 with version 1.0 of our internal SOA system," says Mueller. "We've had to take a strong hand in metering uptake to maintain stability."

Mueller says there's been friction over money and resources, since Web marketing paid for the SOA tier and now it's primarily being used by other groups. "Who pays for it and supports it long term is an unresolved question."

That doesn't sound like a dying technology to us.

From: Susan Fogarty [mailto:susan.fogarty@ubm.com]
Sent: Friday, September 25, 2015 2:54 PM
To: Mukhtar Gohar
Cc: Dr. Anthony F. Norcio
Subject: Re: seeking permission to use an old online article in my PhD dissertation as appendix

Hi Gohar,

Yes, that's fine. Thanks for checking!

Susan Fogarty
Editor in Chief, Network Computing
Interop Las Vegas Content Lead
susan.fogarty@ubm.com
603-583-1306

On Sep 23, 2015, at 4:04 PM, Mukhtar Gohar wrote:

Hi Susan,
Hope you are doing well.
I'm seeking permission for using the following full article as an appendix in my Ph.D. dissertation that I'm currently working on at the IS department at UMBC.

> This story first appeared on InformationWeek NETWORKComputing at http://www.networkcomputing.com/unified-communications/informationweek-analytics-state-of-soa/d/d-id/1076795

Are you the right person for this? If not, could you please guide me to the right person in your organization?

Thanks,
Gohar Mukhtar

12. Appendix H

SOA Practitioners' Guide Part 2: SOA Reference Architecture

By SOA Alliance – Group of SOA Practitioners (09/15/2006)

The relevant section of this Guide is reproduced here (page 15-18) from public domain.

2.2.2 Enterprise SOA Maturity Model

The SOA maturity model helps enterprises develop a roadmap to achieve their target state.

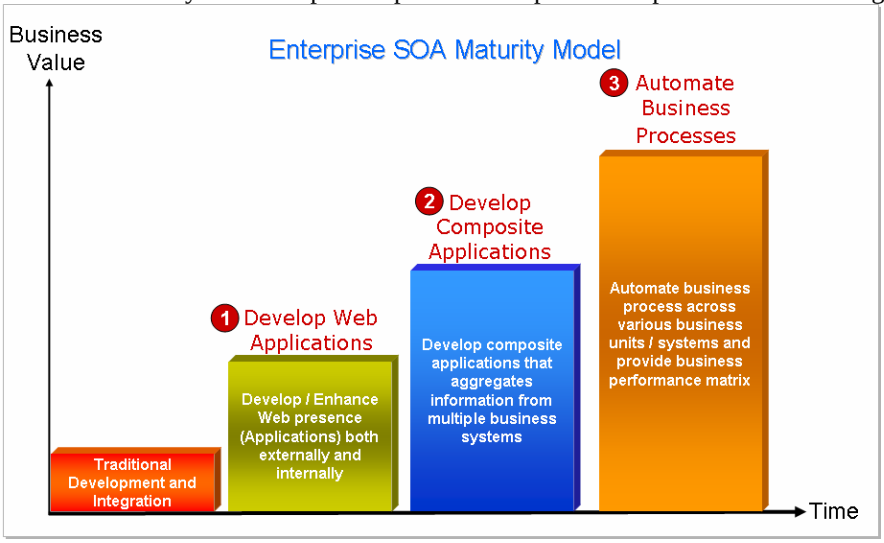


Figure 5: Enterprise SOA Maturity Model

The above diagram illustrates the stages of the enterprise SOA maturity model.

2.2.2.1 Web Application Development Stage

At this stage, teams focus on providing rich client and browser-based business solutions to both internal and external users. They might choose to roll out web-enabled CRM, ERP, or custom applications that support connected and occasionally disconnected operations. In addition, IT organizations typically deploy enterprise-based solutions and services such as content management, search, instant messaging, blogs, Wikis, discussion forums, and white boards.

2.2.2.1.1 Business Requirements

Typically most enterprises would have already deployed external web sites as well as multiple internal web sites and applications to support the diverse needs of each of the business units. The first step is to standardize, share, and integrate these siloed solutions through an infrastructure that provides a common look and feel. This makes it easier for customers, partners, and employees to find the information they are seeking.

During this phase, the team should focus on:

• Unifying user experience on the external site, making it easy for potential users, partners, customers, and analysts to find information they need

• Standardizing the look and feel across all sites (internal and external) as well as across processes and procedures for publishing content

• Creating one my<company name> such as http://my.company.com, site for all employees, contractors, partners, customers to personalize services and content

• Providing secure access to confidential information for all internal and external sites

• Providing a highly reliable, available, and scalable environment

• Enabling the site operations with AJAX to increase performance and user experience.

2.2.2.1.2 Key Challenges

The key challenges for this phase include development of:
• Application support escalation path
• Support for numerous parallel activities
• Leadership and technical quality of team
• Physical environment for development through production, with release management processes and skilled staff resources
• Dedicated production support processes and staffing
• Hosting.

2.2.2.1.3 Exit Criteria
The team can consider this phase complete when:
• External web site is up and running
• Portal front end has been developed for one or more packaged applications
• One or more custom applications is accessible through the portal site
• Most enterprise services have been deployed
• Business users can request information from multiple applications
• Establishment is complete for the program management office (PMO) and LOB governance model for deploying application portals
• Business has confidence in delivery timeline and consistently approaches the program office for all major initiatives.

2.2.2.1.4 Success Factors
This phase is successful if:
• Business involvement at LOB level is high
• Sponsorship/executive oversight has been established for all composite applications
• Web-based applications can be rapidly developed and delivered
• Project management is in place, and the team has leadership and a sense of urgency and direction
• Processes have been standardized across the LOB for development, deployment, and status reporting
• The team has developed identified and created experienced resources.

2.2.2.2 Develop Composite Applications
Composite applications aggregate and provide information and data from a variety of sources and channels, and make them available to internal and external users as appropriate. Enterprise 2.0 services can provide appropriate levels of SLA.

2.2.2.2.1 Business Requirements
The business requirement is for IT to adapt to changing business needs. Several business units may approach IT to develop custom applications, enhance their own branding, increase productivity, or provide additional information to their customers, partners, or employees.
Business requirements may include:
• Branding and exposing multiple applications through the portal
• Accessibility of information from multiple applications
• A web-based desktop for users
• Personalized service based on roles and responsibility of the user
• A single standardized look and feel, which can reduce user training requirements
• Reduced maintenance costs from standardizing on one platform
• Reduced operations and support cost, to enable IT to deploy scarce resources for new functionality.

2.2.2.2.2 Key Challenges
The key challenges for this phase include development of:
• Application support escalation path for shared services
• Support for numerous parallel activities across multiple LOBs
• Governance for shared services
• Leadership and technical quality of team
• Physical environment for development through production, with release management processes and skilled staff resources

• Dedicated production support processes and staffing
• Hosting.

2.2.2.2.3 Exit Criteria
This phase is complete when:
• A Program Management Office (PMO) has been created that spans multiple LOBs, and an enterprise-wide governance model for deploying shared services has been established
• Business has confidence in delivery timelines, and uses the program office for all major initiatives
• Multiple deployed application portals leverage the SOA foundation
• Business units debate integration timeframes for applications or data.

2.2.2.2.4 Success Factors
This phase can be considered a success when:
• Business involvement and sponsorship, including executive oversight, is in place for all composite applications
• The team has developed a rapid development and delivery approach
• Project management has developed leadership, a sense of urgency, and direction
• Processes for development, deployment, and status reporting have been standardized across the enterprise for shared services
• The company has developed experienced resources in agile (parallel development) methodology.

2.2.2.3 Automate Business Processes
This is the stage where the applications, data, and infrastructure help users to perform their roles effectively by providing the right information at the right time. At this stage, the enterprise can start achieving higher ROI by consolidating multiple business systems into a single system. Business organizations should now be ready to abandon their point solutions and transition to the target state of end-to-end business process management.

2.2.2.3.1 Business Requirements
The basic requirements for this phase are as follows:
• Business is interested in standardizing the business process across the enterprise
• Infrastructure is consolidated on standards-based technolog, reducing costs
• Standardized business processes are used globally, but allow for some localization

2.2.2.3.2 Key Challenges
The key challenges for this phase include:
• Accomplishing business and IT transformation
• Establishing appropriate governance and organization models
• Implementing packaged applications for perceived short-term gain.

2.2.2.3.3 Success Factors
This phase is successful when:
• Business involvement and sponsorship and executive oversight enable both business and IT transformation
• A dedicated team focuses on business processes
• Business process is the primary focus for the enterprise
• Loosely coupled business services are assembled to automate business processes and can be recombined to provide new business functionality.

14. Appendix I

BPTrends
BPM Analysis, Opinion and Insight

SOA Maturity Model

By Srikanth Inaganti & Sriram Aravamudan (April 2007)

The most relevant section of this document is reproduced here (page 1-3) from public domain.

The SOA Maturity Model

An SOA maturity model is used to assess the current state of SOA adoption of an organization. The model is used as a yardstick to take stock of as-Is state and develop a transition plan to lead us to the To-Be state. The ultimate aim would be to achieve optimized business services that can nimbly adapt to changing business scenarios.

However, in order to completely gauge the SOA maturity of an organization, it is important to have a multi-point view that encompasses as many aspects of the organization's SOA implementation as possible, to arrive at its true state of SOA maturity. The SOA maturity model proposed in this section takes the following aspects of SOA into consideration to get a full picture of an organization's current level of SOA maturity:

1. Scope of SOA adoption
2. SOA Maturity Level (capabilities of the architecture)
3. SOA Expansion Stages
4. SOA Return On Investment (ROI)
5. SOA Cost Effectiveness and Feasibility

The following diagram is a bird's eye view of the SOA maturity model, depicting the various aspects of SOA maturity.

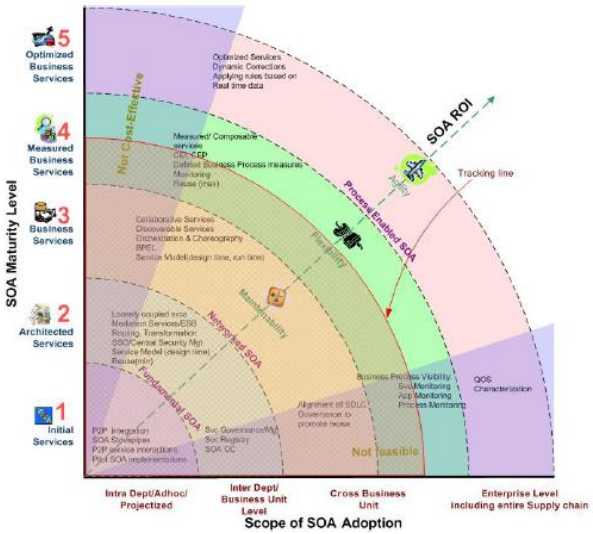


Figure 1. SOA Maturity Model

Salient features of the SOA maturity model

The salient features of the various aspects of SOA maturity described earlier can be summarized as follows. Please refer to the SOA Maturity Model diagram (Figure 1) for further details.

Scope of SOA Adoption: The X- Axis describes the Scope of SOA adoption. As can be seen, it is not a one-to-one mapping between scope of adoption and maturity level. For example Business Unit Level SOA adoption would require a combination of Architected and Business Service maturity in order to achieve effective SOA.

SOA maturity Levels: The Y-Axis shows five levels of SOA maturity along with the key business impact of each level through adding new architectural capabilities with each level of maturity. The SOA characteristics of each maturity level are shown within each level in the concentric quadrant layers along with “Not Cost Effective” and “Not feasible” regions.

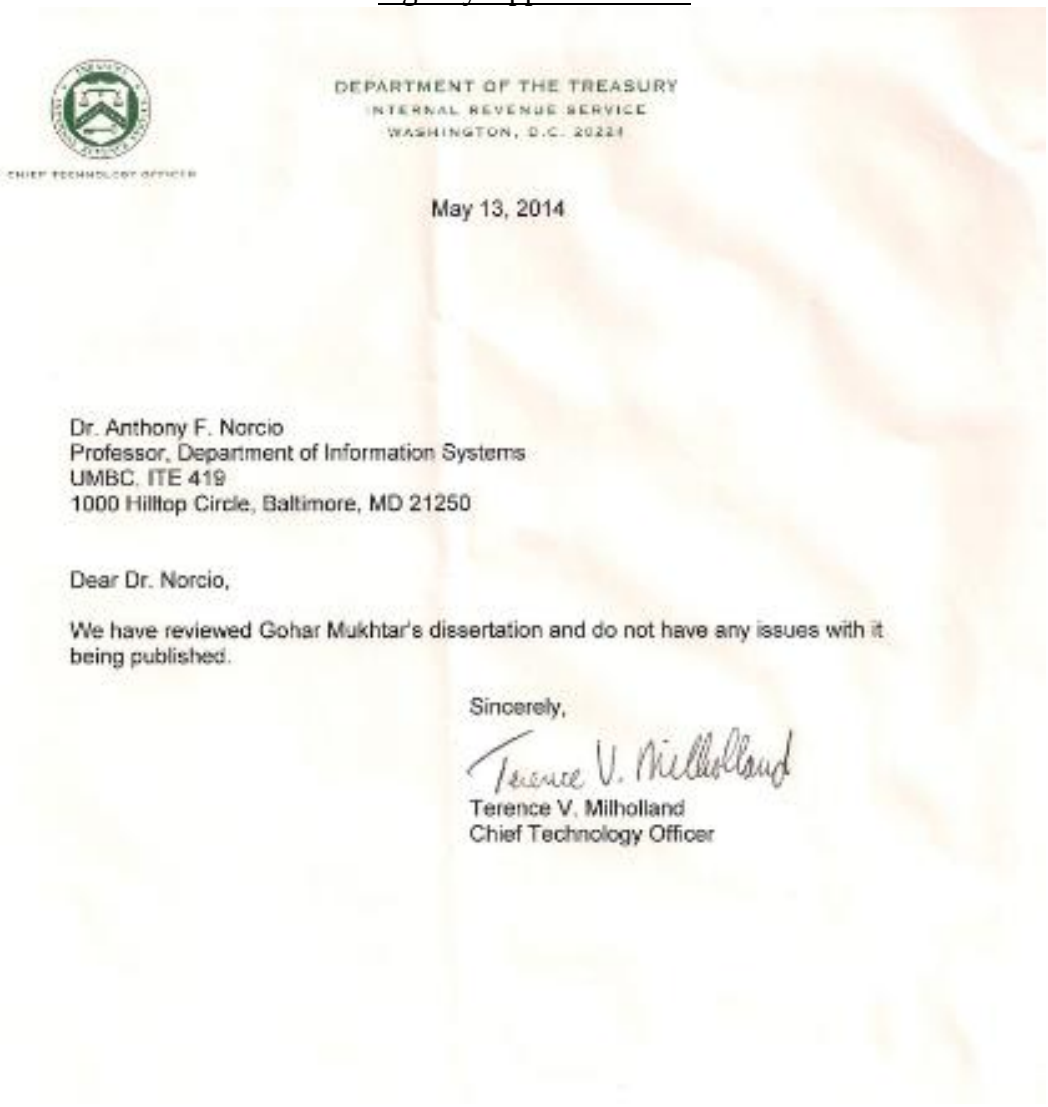
SOA Expansion Stages: Advancement in SOA maturity results in the use of new sets of SOA compliant tools for implementation. This gradual progress in SOA implementation from Fundamental SOA through Networked SOA, culminating in Process oriented SOA has been shown in the quadrant area of the maturity model. Refer [6].

Return on SOA investment: The gradual increase in SOA Return on investment (ROI) with increased maturity level and SOA adoption has been shown in the quadrant section of the model. Increased maintainability is the first ROI, followed by a greater Flexibility, finally resulting in an Agile, Enterprise level system at the highest level of SOA maturity. Refer [6].

SOA Cost Effectiveness and Feasibility: The shaded areas in the maturity model represent the non-cost-effective and infeasible areas of SOA adoption. These areas result when the level of service maturity does not keep up with the degree of SOA adoption. For example, implementing process enabled SOA for intra-department needs may not be cost-effective. Similarly trying to employ fundamental SOA techniques to achieve the goals of enterprise level SOA is not feasible.

15. <u>Appendix J</u>

Agency Approval Letter



DEPARTMENT OF THE TREASURY
INTERNAL REVENUE SERVICE
WASHINGTON, D.C. 20224

CHIEF TECHNOLOGY OFFICER

May 13, 2014

Dr. Anthony F. Norcio
Professor, Department of Information Systems
UMBC. ITE 419
1000 Hilltop Circle, Baltimore, MD 21250

Dear Dr. Norcio,

We have reviewed Gohar Mukhtar's dissertation and do not have any issues with it being published.

Sincerely,

Terence V. Milholland

Terence V. Milholland
Chief Technology Officer

Glossary

The technical terms that are not commonly understood are listed and explained in the table below. Some definitions are adopted from ServiceOrientation.com for standardization.

Term	Meaning and Explanation
Agnostic Logic vs. Non-agnostic Logic	Logic that is sufficiently generic so that it is not specific to (has no knowledge of) a particular parent task is classified as agnostic logic. Because knowledge specific to single purpose tasks is intentionally omitted, agnostic logic is considered multi-purpose. On the flipside, logic that is specific to (contains knowledge of) a single-purpose task is labeled as non-agnostic logic.
Cohesion / Cohesive	The degree to which the elements of a module belong together. It is a measure of how strongly related each piece of functionality expressed by the source code of a software module is. Cohesion is an ordinal type of measurement and is usually described as “high cohesion” or “low cohesion”. Modules with high cohesion tend to be preferable because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability whereas low cohesion is associated with undesirable traits such as being difficult to maintain, difficult to test, difficult to reuse, and even difficult to understand. Cohesion is often contrasted with Coupling. High cohesion often correlates with loose coupling, and vice versa.
Complex Composition	Serious service-oriented solutions are comprised of sophisticated combinations of services. Both in terms of runtime power and design-time complexity, these types of service compositions go well beyond the primitive variation that was more common in the early days of SOA and are therefore referred to as complex compositions.
Component	A component is a unit of logic that exists as a standalone software program as part of a distributed computing architecture. Components can be created with different development tools and programming languages, such as Java and .NET. Component logic can be further exposed via the Web services technology platform through the use of Web service contract-related technologies, such as WSDL, XML schema, and WS-Policy. Although the IT industry places a great deal of

	emphasis on creating services as Web services, it is important to acknowledge that services can be solely constructed from components as long as a meaningful extent of service-orientation is realized.
Contemporary SOA	While primitive SOA represents service-oriented architecture as it can currently be realized, contemporary SOA is a classification that can be used to represent SOA implementations that extend the primitive model using any number of available technologies or products that further the goals associated with service-orientation.
Context (state information type)	Information about a particular service activity (in addition to session data) is qualified with the term context. The larger or more complex a service composition, the more context information will generally need to be managed because more services and inter-service data exchanges will be involved in the corresponding service activity. Context-related information is one of three state information types, the other two being session and business information. State information types are of relevance to the application of the Service Statelessness design principle.
Context Data (context data type)	Information that pertains to what has and is transpiring as part of a current service activity is referred to as context data
Coupling	Coupling refers to a relationship or connection between two things. If two things do not have a connection, they are considered decoupled. If they do have a connection, they are considered coupled, which then raises the question as to what the extent of the coupling is. Something that is coupled to something else may rely on the other thing's existence, which means that the coupling results in a dependency. The extent of coupling therefore may relate to the extent of dependency one thing has on another. This measure of dependency is often communicated with the terms "tight coupling" and "loose coupling," the former indicating a high level of dependency and the latter representing a low degree of dependency.
Data Model Standardization	The standardization of data models used by schemas so as to increase service interoperability.
Design Characteristic	A specific attribute or quality of a body of solution logic that is documented in a design specification and planned to be realized in development. Service-orientation emphasizes the creation of

	very specific design characteristics. Almost every design characteristic is attainable to a certain measure. This means that it is generally not about whether solution logic does or does not have a certain characteristic; it is almost always about the extent to which a characteristic can or should be realized.
Design Paradigm	An approach to designing solution logic which consists of a set of complementary rules or principles that collectively define the overarching approach represented by the paradigm. It provides a set of principles that shape solution logic in certain ways so as to fulfill a specific set of goals.
Design Pattern	A certain way of doing something which provides a proven solution to a common problem.
Design Principle	A generalized, accepted industry practice done to promote common objectives. It proposes a means of accomplishing something based on past experience or industry-wide acceptance. A design principle represents a highly recommended guideline for shaping solution logic in a certain way and with certain goals in mind. These goals are usually associated with establishing one or more specific design characteristics (as a result of applying the principle).
Domain Service Inventory	Domain Inventory is the name of a design pattern. It provides an answer to the challenge when establishing a single enterprise service inventory may be unmanageable for some enterprises, and attempts to do so may jeopardize the success of an SOA adoption as a whole. The solution it advocates is that services can be grouped into manageable, domain-specific service inventories, each of which can be independently standardized, governed, and owned.
Enterprise Service Inventory	Enterprise Inventory is the name of a design pattern which advocates designing of services for multiple solutions within a standardized, enterprise-wide inventory architecture wherein they can be freely and repeatedly recomposed.
Enterprise Service Bus (ESB)	An ESB or Enterprise Service Bus represents an environment designed to foster sophisticated interconnectivity between services. It establishes an intermediate layer of processing that can help overcome common problems associated with reliability, scalability, and communications disparity.
Entity Service	A service with a functional context that is derived from one or more related business entities. Examples of business entities

	include order, client, timesheet, and invoice. Because their functional boundary is based on business entities, Entity services are naturally agnostic to business processes. This allows them to be repeatedly reutilized in support of multiple tasks and business process, positioning them as highly reusable services. Entity services are typically named after their corresponding business entities. For example, it would not be uncommon to label an entity service associated with the invoice business entity as the Invoice service.
Functional Context	Functional Context is the *Cohesion* that binds Service Capabilities together into a service. If a service is looked at as a container of capabilities, the business sense (especially in the Task and Entity services) that suggests grouping related capabilities together is called the Functional Context.
Functional Expression Standardization	Standardized naming conventions can be applied to the delivery of all services so as to ensure the consistent expression of service contexts and capabilities. This approach is referred to as functional expression standardization. Service contracts delivered or extended by different projects and at different times are naturally shaped by the various architects and developers that are required to work with them. The manner in which the service context and the service's individual capabilities are defined and expressed through the contract syntax can therefore vary. Some may use overly verbose conventions, while others may use a terse and highly technical format. Furthermore, the actual terms used to express common or similar capabilities may vary across services. Because services are positioned as enterprise resources, it is fully expected that other project teams will need to discover and interpret the contract in order to be understand how they can use the service. Inconsistencies in how technical service contracts are expressed undermine these efforts by introducing a constant risk of misinterpretation on a technical level. The proliferation of these inconsistencies furthermore places a convoluted face on a service inventory, increasing the effort to effectively navigate various contracts to study possible composition design options.
Federated Endpoint Layer	Federated Endpoint Layer is the name of a design pattern. Federation is an important concept in service-oriented computing. It represents the desired state of the external, consumer-facing perspective of a service inventory, as expressed

	by the collective contracts of all the services in that inventory.
Messaging Metadata	Because messaging does not rely on a persistent connection between service and consumer, it is challenging for a service to gain access to the state data associated with an overall runtime activity. Messaging Metadata is a design pattern that suggests: Message contents can be supplemented with activity-specific metadata that can be interpreted and processed separately at runtime.
Micro SOA	Micro SOA or Micro Services is a software architecture style in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs. These services are small, highly decoupled and focus on doing a small task, facilitating a modular approach to system-building. In SOA terms, the *Functional Context* of such services is precisely and narrowly defined.
Organizational Maturity Levels	From the point at which an organization begins planning for the adoption of SOA and service-orientation up until the time it achieves its planned target state, it can transition through one or more of the common evolutionary levels.
Point-to-point	The term point-to-point originated from the EAI era during which many dedicated integration channels were established between different applications or environments. These integration channels allowed for the exchange of data between specific endpoints. In the world of service-orientation, a point-to-point exchange is comparable to a primitive service activity with a scope limited to a service and a service consumer program.
Runtime Autonomy	The level of control a service has over its processing logic at the time the service is invoked and executing is called runtime autonomy. The Service Autonomy design principle advocates increasing runtime autonomy in order to guarantee the following to service consumers: 1. consistently acceptable runtime execution performance 2. a greater degree of performance reliability 3. the option for it to be isolated in response to specific security, reliability, or performance requirements 4. a greater level of behavioral predictability (especially when concurrently accessed)
Service	A service is a unit of solution logic to which service-orientation

	has been applied to a meaningful extent. It is the application of service-orientation design principles that distinguish a unit of logic as a service compared to units of logic that may exist only as objects or components. Each service is assigned its own distinct functional context and is comprised of a set of capabilities related to this context. Therefore, a service can be considered a container of capabilities associated with a common purpose (or functional context).
Service Activity	The chain of message exchanges carried out in support of the execution of a specific task or business process is referred to as a service activity. There are primitive and complex variations of a service activity. A primitive service activity generally maps to a single data exchange, much like a point-to-point interaction, whereas a complex service activity is usually associated with the message exchanges that occur across a composition of services. Within modern SOA environments, a service activity is generally considered by default to be a complex service activity.
service capability	A service can be seen as a container for a collection of related functions. These functions are called service capabilities and those exposed via a service contract establish a basic API by which the service can be invoked. The term service capability has no implication as to how a service is implemented. Therefore, this term can be especially useful during service modeling stages when the physical design of a service has not yet been determined. Once it is known whether a service exists as a Web service or as a component, the terms “service operation” or “service method” can be used instead.
Service Composition	A service composition is an aggregate of services collectively composed to automate a particular task or business process. To qualify as a composition, at least two participating services plus one composition initiator need to be present. Otherwise, the service interaction only represents a point-to-point exchange.
Service Contract	A service contract is comprised of one or more published documents that express meta information about a service. The fundamental part of a service contract consists of the technical interface which essentially establishes an API into the functionality offered by the service. A service contract can be further comprised of human-readable documents, such as a Service Level Agreement (SLA) that describes additional quality-of-service features, behaviors, and limitations. Within service-orientation, the design of the service contract is of paramount

	importance.
Service Decomposition	Service Decomposition is the name of a design pattern. Overly coarse-grained services can inhibit optimal composition design. An already implemented coarse-grained service can be decomposed into two or more fine-grained services. An increase in fine-grained services naturally leads to larger, more complex service composition designs.
Service Granularity	The overall quantity of functionality encapsulated by a service determines the service granularity. A service's granularity is determined by its functional context which is often derived from one of three common service models. The larger the quantity of related functionality, the coarser the service granularity. Conversely, services with more narrow or targeted functional contexts will tend to have a finer grained level of service granularity. Service granularity represents one of four types of design granularity, the other three being capability, data, and constraint granularity.
Service Inventory	A service inventory is an independently standardized and governed collection of complementary services within a boundary that represents an enterprise or a meaningful segment of an enterprise. When an organization has multiple service inventories, this term is further qualified as domain service inventory.
Service Model	A service model is a classification used to indicate that a service belongs to one of several predefined types based on the nature of the logic it encapsulates, the reuse potential of this logic, and how the service may relate to domains within its enterprise. Three common service models are: Task service, Entity service, and Utility service.
Service Modeling	Modeling services refers to the process and technique of decomposing business process logic into a granular set of individually defined actions that are grouped and organized into service candidates.
Service Normalization	Service Normalization is the name of a design pattern that attempts to answer the question of: How can a service inventory avoid redundant service logic? And advocates that a service inventory needs to be designed with an emphasis on service boundary alignment.

Service-Orientation	Service-orientation is a design paradigm intended for the creation of *Service-oriented solution logic* units that are individually shaped so that they can be collectively and repeatedly utilized in support of the realization of a specific set of strategic goals and benefits associated with SOA and service-oriented computing. Solution logic designed in accordance with service-orientation can be qualified with "service-oriented," and units of service-oriented solution logic are referred to as services.
Service-Oriented Architecture (SOA)	Service-oriented architecture is an architectural model (or style) for building service-oriented solutions with distinct characteristics in support of realizing service-orientation and the strategic goals associated with service-oriented computing.
Service-oriented computing	Service-oriented computing is an umbrella term used to represents a new generation distributed computing platform. As such, it encompasses many things, including its own design paradigm and design principles, design pattern catalogs, pattern languages, a distinct architectural model, and related concepts, technologies, and frameworks.
Service-oriented solution logic	A body of solution logic to which service-orientation has been applied to a meaningful extent is considered "service-oriented." A service represents the most fundamental unit of service-oriented solution logic. There has been a common misperception that the use of Web services technology constitutes a service-oriented solution. It is through service-orientation design principles that solution logic is shaped so that it supports the realization of the strategic goals and benefits associated with SOA and service-oriented computing.
Task Service (also known as Business Service or Business Process Service)	A Task service is a form of business service with a functional context based on a specific business process. As a result, Task services are not generally agnostic and therefore have less reuse potential than other service models. Because Task services tend to represent the end-to-end logic of a business process, they are commonly positioned as service composition controllers, responsible for composing other services (usually Entity and Utility services) to automate their business process. Task services are generally named after the business process they represent. For example a Task service encapsulating logic for the Billing Report process, may be labeled as the Run Billing Report service.

Utility Service	A Utility service is intentionally based on a non-business-centric functional context. It typically encapsulates common, cross-cutting functionality that is useful to many service compositions, but which is not related to or derived from existing business models. As a result, Utility services are commonly agnostic and reusable. Unlike Task and Entity services, the involvement of business analysts or business subject matter experts is generally not required when modeling utility service candidates. Examples of functional contexts that could form the basis of Utility services include notification, event logging, exception handling, and currency conversion.
Web service	A Web service is a body of solution logic that provides a physically decoupled technical contract consisting of a WSDL definition, an XML schema definition, and possibly a WS-Policy definition. This service contract exposes public functions (called operations) and is therefore comparable to a traditional application programming interface (API).

Bibliography

Ahmed, N., & Ahmed, A. (2013). Enabling complexity use case function point on service-oriented architecture. *Computing, Electrical and Electronics Engineering (ICCEEE)* (pp. 535,540). International Conference on Aug. 2013.

Aldris, A., Nugroho, A., Lago, P., & Visser, J. (2013). Measuring the Degree of Service Orientation in Proprietary SOA Systems. *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on 25-28 March 2013* (pp. 233-244). Redwood City: IEEE Computer Society.

Ali, N., Chen, F., & Solis, C. (2012). Modeling Support for Mobile Ambients in Service Oriented Architecture. *Mobile Services (MS)* (pp. 1,8). IEEE First International Conference on 24-29 June 2012.

Andersen, P., Poulsen, B., Trholt, C., & Ostergaard, J. (2009). Using Service Oriented Architecture in a Generic Virtual Power Plant. *Sixth International Conference on 27-29 April 2009* (pp. 1621,1622). Information Technology: New Generations, 2009. ITNG '09.

Arsanjani, A., & Holley, K. (2006). The Service Integration Maturity Model: Achieving Flexibility in the Transformation to SOA. *IEEE International Conference on 18-22 Sep 2006* (p. 515). Chicago, IL: Services Computing, 2006. SCC '06.

Arsenyan, J., & Büyüközkan, G. (2009). Modelling Collaborative Software Development Using Axiomatic Design Principles. *IAENG International Journal Of Computer Science*, 36(3), 234-239.

Atighetchi, M., Webb, J., Loyall, J., & Mayhew, M. (2010). XDDS: A scalable guard-agnostic cross domain discovery service. *Military Communications Conference Oct/Nov 2010* (pp. 1329,1336). MILCOM 2010.

Atkinson, C., & Bostan, P. (2009). Towards a Client-Oriented Model of Types and States in Service-Oriented Development. *Enterprise Distributed Object Computing Conference 1-4 Sept. 2009* (pp. 119,127). IEEE International EDOC '09.

Bahill, T. A., & Botta, R. (2008). Fundamental Principles of Good System Design. *Engineering Management Journal*, 20(4), 9-17.

Barbara, L. (May 1988). Data Abstraction and Hierarchy. *SIGPLAN Notices*, 23,5.

Bloomberg, J. (2013). *The Agile Architecture Revolution.* Hoboken, NJ: John Wiley & Sons, Inc.

Blum, N., Magedanz, T., Schreiner, F., & Wahle, S. (2009). A research infrastructure for SOA-based Service Delivery Frameworks. *TridentCom 2009. 5th International Conference on 6-8 April 2009* (pp. 1,6). Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops.

Booch, G. (1986). Object-oriented development. *IEEE Transactions on Software Enineering, vol. SE-12, no. 2*, 211-221.

Booch, G. (2007). The Economics of Architecture-First. *Software, IEEE (Volume:24, Issue: 5)*, 18-20.

Booch, G. (2008). Architectural Organizational Patterns. *Software, IEEE (Volume:25, Issue: 3)*, 18-19.

Booch, G. (2010). Enterprise Architecture and Technical Architecture. *Software, IEEE (Volume:27, Issue: 2)*, 96.

Breaux, T. (2014). Privacy Requirements in an Age of Increased Sharing. *IEEE Software*, Vol. 31 Issue 5, p24-27.

Bu, H. (2011). Metrics for service granularity in Service Oriented Architecture. *2011 International Conference on 24-26 Dec 2011* (pp. 491,494). Computer Science and Network Technology (ICCSNT).

Buchgeher, G., & Weinreich, R. (2009). Tool Support for Component-Based Software Architectures. *Software Engineering Conference, 1-3 Dec. 2009* (pp. 127,134). APSEC '09. Asia-Pacific.

Cameron, A., Stumptner, M., Nandagopal, N., Mayer, W., & Mansell, T. (2013). A rule-based platform for distributed real-time SOA with application in defence systems. *Military Communications and Information Systems Conference 12-14 Nov. 2013* (pp. 1,7). MilCIS 2013.

Chang, Y.-C., Mazzoleni, P., Mihaila, G., & Cohn, D. (2008). Solving the Service Composition Puzzle. *IEEE International Conference on Services Computing (SCC)* (pp. 387-94). Honolulu, HI: IEEE Computer Society.

Chu, S. (2005). From component-based to service oriented software architecture for healthcare. *HEALTHCOM 2005. Proceedings of 7th International Workshop on 23-25 June 2005* (pp. 96,100). Enterprise networking and Computing in Healthcare Industry, 2005.

Cummins, F. (2009). *Building the Agile Enterprise with SOA, BPM and MBM.* Burlington, MA: Morgan Kaufmann Publishers.

Curbera, F. (2007). Component Contracts in Service-Oriented Architectures. *IEEE Computer Society*, 74-80.

Dahman, K., Charoy, F., & Godart, C. (Aug 29-Sep 2 2011). Towards Consistency Management for a Business-Driven Development of SOA. *Enterprise Distributed Object Computing Conference (EDOC)* (pp. 267,275). 2011 15th IEEE International.

Degeler, V., Gonzalez, L., Leva, M., Shrubsole, P., Bonomi, S., Amft, O., & Lazovik, A. (2013). Service-Oriented Architecture for Smart Environments (Short Paper). *IEEE 6th International Conference on 16-18 Dec. 2013* (pp. 99,104). Service-Oriented Computing and Applications (SOCA).

Dietrich, A., Kirn, S., & Sugumaran, V. (Feb. 2007). A Service-Oriented Architecture for Mass Customization—A Shoe Industry Case Study. *Engineering Management, IEEE Transactions on*, pp.190,204.

Duan, Q., Yan, Y., & Vasilakos, A. V. (2012). A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing. *Network and Service Management, IEEE Transactions on, vol.9, no.4*, pp.373,392.

Dubey, V. K. (2010). *Quality of Service Management of Business Processes in Service Oriented Architectures.* Fairfax, VA: George Mason University.

Eden, A. H., Hirshfeld, Y., & Kazman, R. (2006). Abstraction classes in software design. *IEE Proceedings -- Software*, 153(4), 163-182.

Erl, T. (2008). *SOA: Principles of Service Design.* Upper Saddle River, NJ: Prentice Hall.

Erl, T. (2008). *Web Service Contract Design and Versioning for SOA.* Upper Saddle River, NJ: Prentice Hall.

Faust, J. (2010). An Update on Service Oriented Architecture. *SMPTE Motion Imaging Journal*, 90-91.

Felix, J. M., & Ortin, F. (2014). Aspect-Oriented Programming to Improve Modularity of Object-Oriented Applications. *Journal Of Software*, Vol. 9 Issue 9, p2454-2460.

Fischbach, M., Puschmann, T., & Alt, R. (2011). Towards an interdisciplinary view on service science — The case of the financial services industry. *2011 Federated Conference on 18-21 Sep 2011* (pp. 521,527). Computer Science and Information Systems (FedCSIS).

Fortuna, C., & Mohorcic, M. (Aug 2009). Dynamic composition of services for end-to-end information transport. *Wireless Communications, IEEE*, vol.16, no.4, pp.56,62.

Fowler, M. (2003). Who needs an Architect? *IEEE Software, vol. 20 no. 5*, 2-4.

Garcia-Valls, M., Perez-Palacin, D., & Mirandola, R. (27-30 Jul 2014). Extending the verification capabilities of middleware for reliable distributed self-adaptive systems. *2014 12th IEEE International Conference on* (pp. 164,169). Industrial Informatics (INDIN).

Gerić, S., & Vrcek, N. (22-25 Jun 2009). Prerequisites for successful implementation of Service-Oriented Architecture. *Proceedings of the ITI 2009 31st International Conference on* (pp. 175,180). Cavtat, Croatia: Information Technology Interfaces - ITI 09.

Gomaa, H. (2000). *Designing Concurrent, Distributed, and Real-time Applications with UML*. Addison-Wesley.

Goodall, J. (1998). Learning From the Chimpanzees: A Message Humans Can Understand. *Science 282(5397)*, 2184-2185.

Gottschalk, P., & Solli-Sæther, H. (2009). *E-Government Interoperability and Information Resource Integration: Frameworks for Aligned Development.* Hershey, PA: IGI Global.

Grammatikou, M., Marinos, C., Demchenko, Y., Lopez, D., Dombek, K., & Jofre, J. (2011). GEMBus as a Service Oriented Platform for Cloud-Based Composable Services. *2011 IEEE Third International Conference on Nov 29 - Dec 01 2011* (pp. 666,671). Cloud Computing Technology and Science (CloudCom).

Gringinger, E., Trausmuth, G., Balaban, A., Jahn, J., & Milchrahm, H. (2012). Experience report on successful demonstration of SWIM by three industry partners. *Integrated Communications, Navigation and Surveillance Conference 24-26 April 2012* (pp. G6-1,G6-8). ICNS.

Gu, C., & Zhang, X. (29-31 July 2010). An SOA Based Enterprise Application Integration Approach. *2010 Third International Symposium on* (pp. 324,327). Electronic Commerce and Security (ISECS).

Guinard, D. D., Trifa, V. V., Karnouskos, S. S., Spiess, P., & Savio, D. (2010.3). Interacting with the SOA-based internet of things: discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions On Services Computing*, 3(3), 223-235.

Haoyu, W., & Haili, Z. (August 2012). Basic Design Principles in Software Engineering. *Computational and Information Sciences (ICCIS)* (pp. pp.1251,1254). Fourth International Conference on.

Hassan, Q. F. (2009). Aspects of SOA: An Entry Point for Starters. *Annals.Computer Science Series, 7(2)*, 125-142.

High, R., Krishnan, G., & Sanchez, M. (2008). Creating and maintaining coherency in loosely coupled systems. *IBM Systems Journal , vol.47, no.3*, pp.357,376.

Huhns, M., & Singh, M. P. (2005). Service-oriented computing: key concepts and principles. *IEEE Internet Computing*, 9(1), 75-81.

Hunter, P. (October 11 2008). Across the generation. *Engineering & Technology, vol.3, no.17,* pp.56,59.

Hutchinson, J., Kotonya, G., Walkerdine, J., Sawyer, P., Dobson, G., & Onditi, V. (Jan.-Feb. 2008). Migrating to SOAs by Way of Hybrid Systems. *IT Professional , vol.10, no.1*, pp.34,42.

Jain, B., & Kumar, R. (2007). Anatomy of Service-oriented Architecture. *Institute of Chartered Financial Analysts of India (ICFAI) University Press, Journal Of Information Technology*, 67-80.

Kannan, K., Bhamidipaty, A., & Narendra, N. (2011). Design Time Validation of Service Orientation Principles using Design Diagrams. *2011 Annual SRII Global Conference* (pp. 795-804). IEEE Computer Society.

Karhikeyan, T., & Geetha, J. (25-27 April 2012). A metrics suite and fuzzy model for measuring coupling in Service Oriented Architecture. *2012 International Conference on* (pp. 254,259). Recent Advances in Computing and Software Systems (RACSS).

Khoshkbarforoushha, A., Tabein, R., Jamshidi, P., & Shams, F. (5-10 July 2010). Towards a Metrics Suite for Measuring Composite Service Granularity Level

Appropriateness. *2010 6th World Congress on* (pp. 245,252). Services (SERVICES-1).

Kral, J., & Zemlicka, M. (2009). Popular SOA Antipatterns. *IEEE Computer Society - ComputationWorld*, 271-276.

Kumar, R., Haber, A., Yazidi, A., & Reichert, F. (2010). Towards a relation oriented service architecture. *2010 Second International Conference on 5-9 Jan. 2010* (pp. 1,8). Communication Systems and Networks (COMSNETS).

Kumaran, S., Bishop, P., Chao, T., Dhoolia, P., Jain, P., Jaluka, R., . . . Nigam, A. (2007). Using a model-driven transformational approach and service-oriented architecture for service delivery management. *IBM Systems Journal, vol.46, no.3*, pp.513,529.

Legner, C., & Vogel, T. (9-13 July 2007). Design Principles for B2B Services - An Evaluation of Two Alternative Service Designs. *Services Computing* (pp. 372,379). SCC 2007, IEEE International Conference on.

Lewis, G., Morris, E., Simanta, S., & Smith, D. (Jan-Feb 2011). Service Orientation and Systems of Systems. *IEEE Software vol.28, no.1*, pp.58,63.

Li, H., & Wu, Z. (2009). Research on Distributed Architecture Based on SOA. *International Conference on Communication Software and Networks* (pp. 670-674). IEEE Computer Society.

Lund, K., Eggen, A., Hadzic, D., Hafsoe, T., & Johnsen, F. (October 2007). Using web services to realize service oriented architecture in military communication networks. *IEEE Communications Magazine*, vol.45, no.10, pp.47,53.

Mamaghani, N., Mousavi, F., Hakamizadeh, F., & Sadeghi, M. (23-25 June 2010). Proposed combined framework of SOA and RUP. *2010 3rd International*

Conference on (pp. 346,351). Information Sciences and Interaction Sciences (ICIS).

Manes, A. T. (2013, 11 04). Solution Path: Executing Your SOA Initiative. *Gartner Technical Professional Advise G00254072*. Stanford, CT: Gartner, Inc.

Marks, E. (2006, 08). A Test of Maturity. *Managing Automation*, pp. Vol. 21 Issue 8, p71-71, 1p.

Martin, R. C. (2003). *Agile Software Development, Principles, Patterns, and Practices.* Upper Saddle River, N.J.: Prentice Hall.

Mauro, C., Leimeister, J. M., & Krcmar, H. (2010). Service Oriented Device Integration – An Analysis of SOA Design Patterns. *2010 43rd Hawaii International Conference on System Sciences (HICSS)* (pp. 1-10). Honolulu, HI: IEEE Computer Society.

Mazzarolo, C., Martins, V., Toffanello, A., & Puttini, R. (Jan. 2015). A Method for SOA Maturity Assessment and Improvement. *Latin America Transactions, IEEE (Revista IEEE America Latina) , vol.13, no.1,* 204,213.

Meye, B. (Oct 1992). Applying "Design by Contract". *IEEE Computer, 25, 10,* 40–51.

Meyer, B. (2007). Contract-driven development. *Proceedings of the 10th international conference on Fundamental approaches to software engineering* (p. 11). Berlin, Heidelberg: Springer-Verlag.

Microsoft Services. (2007). Assessment and Roadmap for Service Oriented Architecture. Microsoft Corporation.

Miller, G. A. (1955). The Magical Number Seven, Plus or Minus Two Some Limits on Our Capacity for Processing Information. *Psychological Review Vol. 101, No. 2*, 343-352.

Mukhtar, G. (11/2011). Demystifying Service-Oriented Architecture Part-I: Promised Goals and Benefit. *unpublished*.

Mukhtar, G. (12/2011). Demystifying SOA Part-II: Principles of Service Design. *unpublished*.

Mukhtar, G. (2011). Demystifying Service-Oriented Architecture Part-III: Commonly Used Design Patterns. *unpublished*.

Murer, S., & Hagen, C. (Nov-Dec 2014). Fifteen Years of Service-Oriented Architecture at Credit Suisse. *IEEE Software, vol.31, no.6*, 9,15.

Nath, S. (2012). Web services: Design Choices for Space Ground System Integration. *Military Communications Conference, Oct 29 - Nov1 2012* (pp. 1,6). MILCOM.

Nayak, N., & Nigam, A. (23-26 July 2007). Modeling Business Services for Implementing on Global Business Services Delivery Platforms. *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on* (pp. 577-583). Tokyo: IEEE.

Ning, F., Zhou, W., Zhang, F., Yin, Q., & Ni, X. (2011). The architecture of cloud manufacturing and its key technologies research. *2011 IEEE International Conference on 15-17 Sept. 2011* (pp. 259,263). Cloud Computing and Intelligence Systems (CCIS).

Offermann, P., Hoffmann, M., & Bub, U. (2009). Benefits of SOA: Evaluation of an implemented scenario against alternative architectures. *13th Enterprise Distributed Object Computing Conference Workshops 1-4 Sept. 2009* (pp. 352,359). EDOCW 2009.

Ollinger, L., Zuhlke, D., Theorin, A., & Johnsson, C. (2013). A reference architecture for service-oriented control procedures and its implementation with SysML and Grafchart. *IEEE 18th Conference on10-13 Sept. 2013* (pp. 1,8). Emerging Technologies & Factory Automation (ETFA).

Papazoglou, M. P., & van den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal, Volume 16, Issue 3*, 389-415.

Parlanti, D., Paganelli, F., & Giuli, D. (June 2011). A Service-Oriented Approach for Network-Centric Data Integration and Its Application to Maritime Surveillance. *IEEE Systems Journal*, vol.5, no.2, pp.164,175.

Parnas, D. L., & Morris, R. (1972). On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM, Dec 1972, Vol. 15 Issue 12*, 1053-1058.

Rathfelder, C., & Groenda, H. (2008). iSOAMM: An Independent SOA Maturity Model. *Distributed Applications and Interoperable Systems* (pp. 1-15). Berlin Germany: Springer Berlin Heidelberg.

Rostampour, A., Kazemi, A., Zamiri, A., Haghighi, H., & Shams, F. (2011). A metric suite for measuring service alignment with business agility. *CSI International*

Symposium on 15-16 June 2011 (pp. 121,128). Computer Science and Software Engineering (CSSE).

Ruz, C., Baude, F., Sauvan, B., Mos, A., & Boulze, A. (2011). Flexible SOA Lifecycle on the Cloud Using SCA. *15th IEEE International Aug. 29 2011-Sept. 2 2011* (pp. 275,282). Enterprise Distributed Object Computing Conference Workshops (EDOCW).

Saleh, I., Kulczycki, G., & Blake, B. (Sep/Oct 2009). Demystifying Data-Centric Web Services. *IEEE Internet Computing*, 86-90.

Schultz, A. P., Fricke, E., & Igenbergs, E. (July 2000). Enabling Changes in Systems Throughout the Entire Life-Cycle - Key To Success? *Proceedings of the 10th Annual International Symposium of the International Council on Systems Engineering (INCOSE)*, (pp. 599-607).

Seaman, C. B. (1999). Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering, VOL. 25, NO. 4*, 557 - 572.

Selmeci, R., & Rozinajova, V. (2012). One approach to partial formalization of SOA design patterns using production rules. *2012 Federated Conference on 9-12 Sept. 2012* (pp. 1381,1384). Computer Science and Information Systems (FedCSIS).

Service Oriented Architecture Reference Model Technical Committee. (2012, 12 04). Reference Architecture Foundation for Service Oriented Architecture Version 1.0. *Committee Specification 01*. Organization for the Advancement of Structured Information Standards (OASIS).

Simonelis, A. (2004). Design Principles Are Where You Find Them. *Communications Of The ACM*, 47(11), 11.

Sonic Software Corporation, AmberPoint Inc., BearingPoint, Inc., Systinet Corporation. (2005). A New Service-Oriented Architecture (SOA) Maturity Model. Object Management Group (OMG®).

Stachtiari, E., Vesyropoulos, N., Kourouleas, G., Georgiadis, C., & Katsaros, P. (2014). Correct-by-Construction Web Service Architecture. *IEEE 8th International Symposium on 7-11 April 2014* (pp. 47,58). Service Oriented System Engineering (SOSE).

Stal, M. (Mar-Apr 2006). Using architectural patterns and blueprints for service-oriented architecture. *IEEE Software*, vol.23, no.2, pp.54,61.

Sud, M. (2010). *Sensitivity in Service Design for the Development of SOA Based Systems.* Windsor, Ontario, Canada: University of Windsor, Ontario, Canada.

Sulong, M. S.-A. (2013). The Importance of Considering Information Quality in the Implementation of Service-Oriented Architecture Initiatives. *IEEE 10th International Conference on Services Computing* (pp. 372-383). Santa Clara, CA: IEEE Computer Society.

Takdir, & Kistijantoro, A. (24-25 Sept. 2014). Multi-layer SOA implementation pattern with service and data proxies for distributed data-intensive application system. *2014 International Conference on* (pp. 37,41). ICT For Smart Society (ICISS).

Taylor, S., & Bogdan, R. (1984). *Introduction to Qualitative Research Methods.* New York: John Wiley & Sons.

The Open Group. (2009). *SOA Source Book.* Zaltbommel: Van Haren Publishing. Retrieved from The Open Group.

The Open Group. (2011). The Open Group Service Integration Maturity Model (OSIMM), Version 2. *Technical Standard*. The Open Group. Retrieved from The Open Group.

Valipour, M., Amirzafari, B., Maleki, K., & Daneshpour, N. (2009). A brief survey of software architecture concepts and service oriented architecture. *2nd IEEE International Conference on Computer Science and Information Technology 8-11 Aug 2009* (pp. 34,38). ICCSIT.

Vrba, P., Marik, V., Siano, P., Leitao, P., Zhabelova, G., Vyatkin, V., & Strasser, T. (Aug 2014). A Review of Agent and Service-Oriented Concepts Applied to Intelligent Energy Systems. *Industrial Informatics, IEEE Transactions on*, vol.10, no.3, pp.1890,1903.

Wagner, M., Zobel, D., & Meroth, A. (2014). SODA: Service-Oriented Architecture for Runtime Adaptive Driver Assistance Systems. *IEEE 17th International Symposium on 10-12 June 2014* (pp. 150,157). Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC).

Walsh, J. J. (1920). Man As Tool and Weapon Maker. *America, 23(21),* 485-486.

Ward, M. (2006). Using Software Design Methods in CALL. *Computer Assisted Language Learning*, 19(2-3), 129-147.

Welke, R., Hirschheim, R., & Schwarz, A. (2011). Service-Oriented Architecture Maturity. *Computer, 44(2),* 61-67.

Wirfs-Brock, R. J. (2009). Principles in Practice. *IEEE Software*, 26(4), 11-12.

Xiao-Jun, W. (19-20 Dec. 2009). Metrics for Evaluating Coupling and Service Granularity in Service Oriented Architecture. *International Conference on* (pp. 1,4). Information Engineering and Computer Science ICIEC.

Xu, Y.-c., Wang, L., Xie, J., & Xia, G.-p. (19-20 Dec. 2009). Research and Realization for Simulation System of Rock-Fill Dams Based on Service-Oriented-Architecture. *ICIECS 2009. International Conference on* (pp. 1,4). Information Engineering and Computer Science, 2009.

Zhou, N., & Zhang, L.-J. (2009). Analytic architecture assessment in SOA solution design and its engineering application. *2009 IEEE International Conference on Web Services (ICWS)* (pp. 807-814). Los Angeles, CA: IEEE Computer Society.

Zimmermann, O. (2011). Architectural Decisions as Reusable Design Assets. *IEEE Software, 28(1),* 64-69.