View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AIS Electronic Library (AISeL)

Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2009 Proceedings

Americas Conference on Information Systems (AMCIS)

2009

Service-Oriented Software Development

Mark Keith
Arizona State University, mark.keith@asu.edu

Haluk Demirkan
Arizona State University, haluk@uw.edu

Michael Goul
Arizona State University, michael.goul@asu.edu

Follow this and additional works at: http://aisel.aisnet.org/amcis2009

Recommended Citation

Keith, Mark; Demirkan, Haluk; and Goul, Michael, "Service-Oriented Software Development" (2009). *AMCIS 2009 Proceedings*. 100.
http://aisel.aisnet.org/amcis2009/100

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2009 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

المنارة للاستشارات

www.manaraa.com

Service-Oriented Software Development

Mark Keith
Arizona State University
Mark.Keith@asu.edu

Haluk Demirkan
Arizona State University
Haluk.Demirkan@asu.edu

Michael Goul
Arizona State University
Michael.Goul@asu.edu

ABSTRACT

This paper describes a methodology for planning and executing software development projects based on the service-oriented paradigm called Service-Oriented Software Development (SOSD). This does not refer to a methodology for developing software services or service-oriented architectures. Rather, it is a method for managing the *process* of software development in a service-oriented approach in which the actions performed by individuals and groups are modeled as "services" which can be choreographed and orchestrated. SOSD has been adopted informally by many organizations in various forms and it runs somewhat contrary to some of the recent trends toward agile development methodologies. Interestingly, it performs well in certain situations where agile methods tend to break down. In addition to describing the basics of SOSD and its theoretical underpinnings, we outline its benefits and potential shortcomings. As evidence, project data is provided from a Fortune 500 company which has gradually adopted SOSD over the last two years.

Keywords (Required)

Service-oriented, software development, agile, project management, SOA.

INTRODUCTION

One of the most dominant themes in the IT project management literature is how to reduce project risk and improve project performance (Schmidt, Lyytinen, Keil, & Cule, 2001). Because of the difficulty in planning large-scale software development projects, inadequateness of requirements analyses, and the constant changes in scope, agile software development (ASD) methodologies (e.g. XP, SCRUM, etc.) have gained increased interest among researchers and practitioners (Highsmith & Cockburn, 2001; Dyba & Dingsøyr, 2008). To combat the high costs of change that traditional methods (e.g. waterfall) can have, agile methods implement short iterations of planning, development, testing, and feedback. In essence, these project teams are planning on changes taking place throughout the project and minimize the time and cost wasted in planning every aspect of the project through to completion. As a result, ASDs have the potential to minimize the project loss resulting from increased risk and uncertainty created by changing project demands.

While ASDs have certainly been successful in many organizations (e.g. Sutherland, Viktorov, Blount, & Puntikov, 2007), they rely on certain assumptions. For example, less planning upfront means that more of the task interdependencies must be managed during project execution in the form of ad-hoc communication among team memebers referred to as "mutual adjustment" (Thompson, 1967). To adequately facilitate this "on the fly" coordination, project teams generally need strong knowledge sharing networks and high levels of expertise (Turk, France, & Rumpe, 2005). But what if there is a high degree of turnover in project teams or subject matter experts leave the company? Can agility amidst risk and uncertainty still be achieved if the ground is not fertile for ASD methodologies?

In recent years, the concept of service-orientation as a paradigm for technical infrastructures, application design, resource allocation, and business processes has gained momentum (Chesbrough & Spohrer, 2006). In large part, this is because of the potential it offers for greater organizational agility (Erl, 2005). In this paper, we describe a service-oriented approach to managing software development projects which, like ASD, can improve software process agility, though in a manner different from ASD methodologies. Both ASD methodologies and the approach presented here are intended to better manage project risk and uncertainty. But, whereas ASD embraces project uncertainty by planning for change throughout the project life cycle, the method outlined in this paper is designed to reduce uncertainty upfront and have an infrastructure of readily-

www.manaraa.com

available and easily-shared knoweldge and experience available as changes arise. Simply put, it improves project plans upfront to reduced the changing of plans later.

While this concept of service-oriented software development is being adopted informally in certain capacities in some organizations, there is currently no defined methdology of this type. Therefore, this research takes a design science approach based on the set of research principles outlined in the seminal work by Hevner, March, Park, and Ram (2004). We develop a formal, high-level description and theoretical explanation of the methodology which can easily be generalized to a large population of software development units followed by an outline of the benefits and potential risks. Next, we present some brief practical evidence of it's implementation at the software development unit of a Fortune 500 company. Finally, we discuss some implications for future research and practice.

SERVICE-ORIENTATION

At its most fundamental level, the service-oriented paradigm is simply an "approach for separating concerns" (Erl, 2005, pp. 32; Foster, 2005). This means the logic required to solve complex problems can be broken down into smaller and more manageable components which each provide a specific service to the larger problem domain. In this generic sense, service-orientation can take place at practically any level of abstraction and for any problem-solving scenario where decomposition is possible. However, until more recently, the literature has typically used the term "service-oriented" to refer to breaking down complex software applications into smaller and more manageable components such as Web Services (Newcomer, 2003).

Service-orientation is based on the concept of modularity which refers to "the degree to which a system's components can be separated and recombined, and… both the tightness of coupling between components and the degree to which the 'rules' or the system architecture enable the mixing and matching of components" (Schilling, 2000, p. 312). The term "modular system" can refer to several levels. In the strategic management and organization literature, the modularity concept is applied to product architectures, business processes, and organizational structures which face environments of increasing complexity and uncertainty (Hoetker, 2006).

One of the basic premises of this line of research is that modularity in the product architecture allows for greater agility when the need for change arises. Modular products allow easier recombination of inputs and outputs. For example, modular software is developed in components with standardized interfaces (Parnas, 1985). The division of software into modularized components takes place along natural "break points" in the application where there is little interdependence between modules. By doing so, parts of the application can be easily changed with minimal interference elsewhere.

However, there is an implicit tradeoff regarding the degree of modularity. As products or organizations become increasingly modular, they also become increasingly complex because of the greater coordination required to manage a larger number of elements (Langlois, 2002). As such, Baldwin and Clark (1997) argue that, to be effective, modular systems require:

1. An architecture, which specifies all modules and their respective functions.
2. Interfaces for module interaction, coordination, and communication.
3. Standards for testing the module's conformity to design rules.

Baldwin and Clark's (1997) three requirements for effective modular systems are met by service-oriented architectures (SOA). OASIS defines SOA as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains (MacKenzie, 2007). It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. The "capabilities" referred to are the elements produced by modularization viewed in terms of the capabilities, or services, they can provide to the overall process or product. Typically, SOA includes a technical architecture used to store and categorize the services (e.g. Web Services), thus meeting Requirement 1. In compliance with Requirement 2, SOAs also define the interfaces that allow service interaction, coordination, and communication. For example, WSDL files are used to define Web Service interfaces, while SOAP is the XML-based messaging system used by Web Services to communicate. Finally, a variety of standardized service quality measures are being developed which allows organizations to ensure that the services they use conform to predefined quality metrics (Sheth, Verma, & Gomadam, 2006), thus meeting Requirement 3 for effective modularized systems.

Traditionally, SOAs help the software development units of organizations to achieve software product modularity. But their use is beginning to extend beyond the management of software or technical services only. Just as Web Services are resources which perform services for an application, individuals and groups can also be viewed as resources which perform services in a business process (Bieberstein, Bose, Walker, & Lynch, 2005). And these human resources can be described, coordinated, and invoked as services from an SOA using standards for interfacing, description, and discovery (Sheth et al., 2006).

www.manaraa.com

Therefore, the OASIS definition describes SOA as a "paradigm" for managing "capabilities" because it can be used to manage any type of distributed capability – whether it be technical (e.g. Web Service), human, or organizational (MacKenzie, Laskey, McCabe, Brown, & Metz, 2006). The defining characteristics of SOA are that it facilitates the organization and utilization of services and that those services may be distributed across different ownership domains. For example, just as the Web Services used by a single application may reside on distributed web servers both within and without an organization, the human services provided to develop software may come from internal personnel and external contractors and consultants.

Using this larger context of SOA for human services, the next section conceptualizes how the process of software development (and not just the resulting software) can become service-oriented and managed through the use of SOAs.

Service-Oriented Software Development

Business process agility (BPA) refers to the speed and flexibility with which firms can dynamically modify or reconfigure their business processes to accommodate the demands of their environment (Raschke & David, 2005). As discussed, organizations are adopting SOAs in order to improve agility. By composing their software architecture of reusable and loosely-coupled Web Services, organizations can readjust their software architecture with greater speed and flexibility as the business processes change (Erl, 2005). Now, as techniques for managing human services under an SOA paradigm in addition to Web Services are emerging, a greater degree of reuse and agility may be achieved (Bieberstein et al., 2005). By managing both technical and human services through an SOA, both the software architecture and the business processes it supports can become service-oriented.

Software projects can be managed as a set of tasks or "services" performed by individuals and groups. We refer to this as service-oriented software development (SOSD). In its most basic form, the SOSD methodology entails: 1) that the software development process is modularized into the natural services (or human-performed activities) it consists of, and 2) the use of an SOA. The modularization task is basically a form of business process modeling except that besides creating the "AS-IS" and "TO-BE" models of the business process being affected by the software, the software development process itself is also modeled. This is one reason why business process management and SOA are viewed as very synergistic (Oracle, 2008). The SOA, at a minimum, accomplishes two basic tasks. First, it consists of a few repositories – one for storing service descriptions (both human and technical), one for storing the service patterns which comprise business processes, one for storing the resources used to perform services (e.g. a list of employees, contractors, and consultants and the services they can each provide), and one for storing historical patterns of resources allocated to services. Second, the SOA offers a mapping engine which is used to help managers quickly and dynamically assigned resources to services. These requirements are more fully understood in the following hypothetical example.

Consider a typical project-based software development environment where customers (either external or "business side") make requests for new applications or components to add to an existing software. Each project is assigned to a particular project manager (PM) who spends the next several months working entirely on that project. As part of the planning phase, a PM will first search the SOA service pattern repository for an appropriate, or "best-fitting" human service pattern (See Figure 1). A service pattern is simply a template which outlines the conceptual services needed to complete a particular type of task[1]. Figure 1 represents a hypothetical service pattern which the PM might select as the closest match to the needs to the newly assigned project. The boxes labeled "Service A," "Service B," "Service C," and "Service D" represent less-granular or high-level services. Each of these services is composed of its own pattern of sub-services (A_a, A_b, A_c, B_a, B_{b1}, B_{b2}, B_{b3}, B_c, C_a, C_b, C_c, D_a, D_b). Some sub-services are composed of their own pattern of sub-sub-services (B_{bb}, B_{bc}). Each level (service, sub-service, or sub-sub-service) possesses the same generic service properties and differs only in terms of granularity.

The sub-service pattern that comprises the "development" service (Service B) demonstrates how each activity may have multiple potential service patterns. In the example in Figure 1, three components are produced in the development process (characterized by sub-services B_{b1}, B_{b2}, and B_{b3}) each with a unique pattern of sub-sub-services. Sub-service B_{b1} is performed by developing the component code first followed by the component database. Sub-service B_{b2} does not need a database component. Sub-service B_{b3} can be performed by developing both the code and database simultaneously. In practice, actual service patterns will likely be much more granular and define services at a more detailed level. However, the example demonstrates how each service may have multiple potential sub-service patterns just as each software development project may have multiple potential service patterns. Once an adequately-fitting service pattern is found, the PM can then go through the process of selecting and allocating the resources needed to complete the service pattern. This is accomplished by

[1] It is important to note that each process may have multiple legitimate service patterns. The "best" pattern in this scenario may be determined by factors such as the availability of resources, need for low cost versus quick delivery, etc.

www.manaraa.com

searching the SOA repository for resource allocation patterns linked to that service pattern. Resource allocation patterns are a historical record of the actual patterns of software, hardware, and human services assembled to accomplish prior projects.

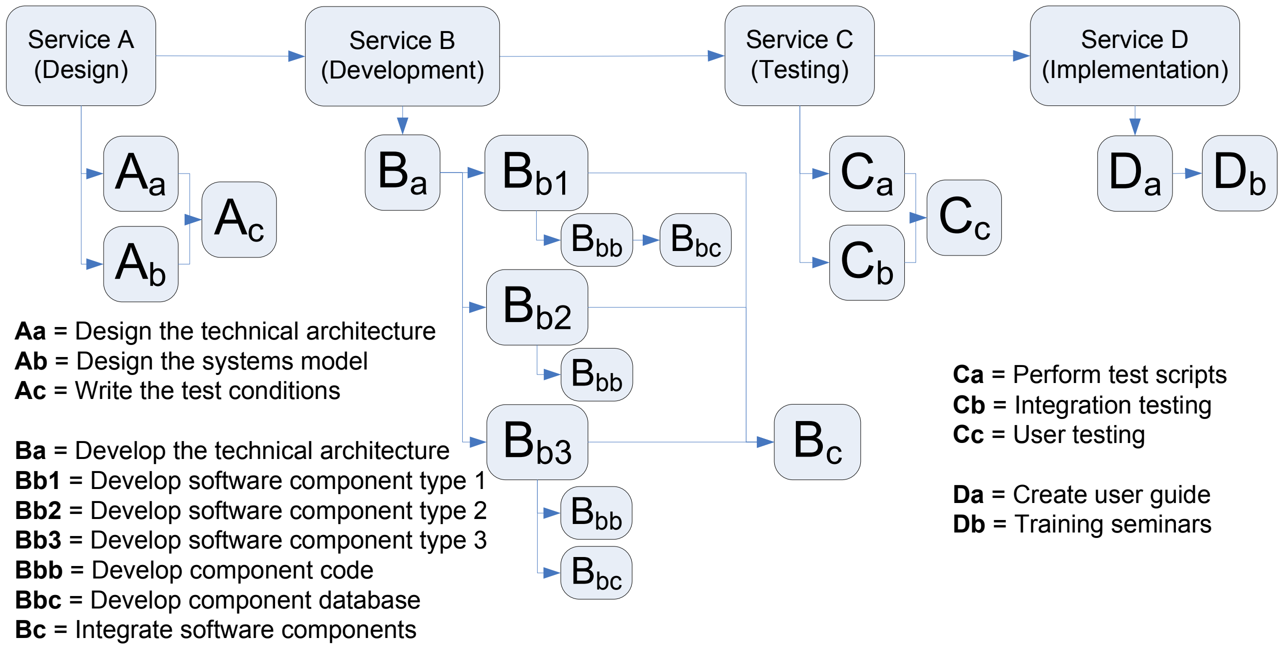
Service A (Design)
Service B (Development)
Service C (Testing)
Service D (Implementation)
Aa Ab Ac Ba Bb1 Bbb Bbc Bb2 Bbb Bb3 Bbb Bbc Bc Ca Cb Cc Da Db
Aa = Design the technical architecture
Ab = Design the systems model
Ac = Write the test conditions
Ba = Develop the technical architecture
Bb1 = Develop software component type 1
Bb2 = Develop software component type 2
Bb3 = Develop software component type 3
Bbb = Develop component code
Bbc = Develop component database
Bc = Integrate software components
Ca = Perform test scripts
Cb = Integration testing
Cc = User testing
Da = Create user guide
Db = Training seminars

Figure 1: Service Choreography Example

Figure 2 visualizes a hypothetical resource allocation pattern for the development service only (Service B) from Figure 1. The numbers on each resource represent unique identifiers. The example in Figure 2 demonstrates how some people may be assigned to multiple services (e.g. Person 6 works on both sub-sub-service B_{bc} and sub-service B_c). Each service, sub-service, or sub-sub-services has its own set of patterns for how it might be accomplished depending on which resources are available.

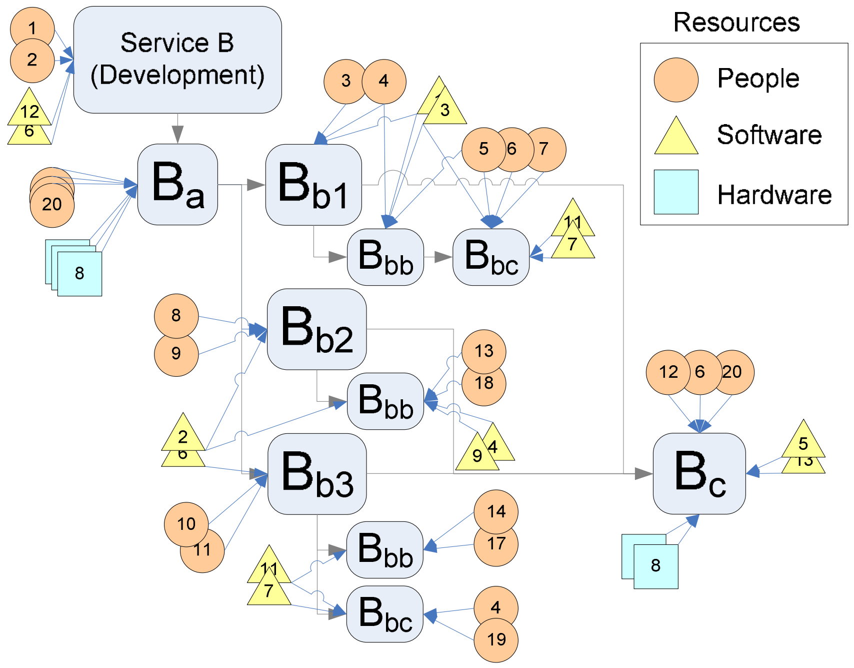

Figure 2. Service Orchestration Example

www.manaraa.com

Besides providing a repository for service patterns and resource allocation patterns, the SOA uses a mapping engine to dynamically assign resources to services. In this way, the SOA can speed the task of resource allocation and reduce errors and uncertainty. For example, the PM may begin by selecting a particular resource allocation pattern only to discover that some individuals have scheduling conflicts with other project assignments. The SOA mapping engine can search the repository of service providers and suggest alternatives. In addition, some of the service providers suggested by the SOA may external contractors or consultants if internal personnel are not be available (See Figure 3). Thus, SOA can facilitate more targeted outsourcing (Erl, 2005). In addition, outsourcing may be further facilitated if the SOSD organization provides its contractors and consultants access to its SOA repository of service definitions.

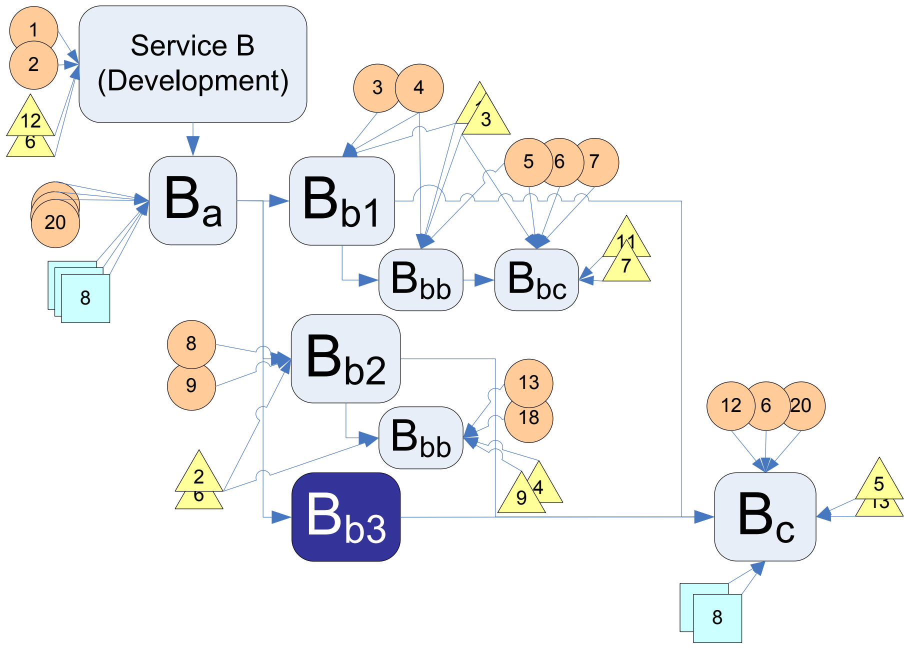

Figure 3. Service Outsourcing Example

In Figure 3, sub-service Bb3 is colored dark blue to represent an outsourced service. Notice that there are no resources (human or technical) attached to that service because the SOSD organization does need to know how the outsourced provider is performing the service. When service provider availability is established, the SOA can schedule the time needed from each employee in its own "master" schedule. The new resource allocation pattern is recorded in the SOA repository and project work begins. These human services are "invoked" by reserving their project time through the SOA.

In summary, the SOSD methodology is can reduce project uncertainty upfront because it enables the reuse of prior project knowledge and experience. This is made possible by implementing a repository for storing standardized process service patterns and the actual historical executions of those patterns. As mentioned above, it extends previous conceptions of the agility and reuse capabilities offered by traditional SOA by broadening its usage to the software development process itself. This means that PMs can spend less time upfront in planning, analysis, and resource allocation.

It might be argued that SOSD does not solve one of the fundamental reasons that organizations adopt ASDs – that is, because changes to scope and requirements take place after the initial project kick-off. What SOSD does enable in this case is the ability to "plug-n-play" human services just as you could substitute software services in an application. In other words, because the services have already been modeled in great detail, it will be easier to identify the exact activities that need to be added, dropped, modified, or outsourced to meet the new requirements. In addition, the service pattern repository can be search at varying levels of granularity – meaning that a PM could return to the repository to select a new sub-pattern or sub-sub-pattern only. They would not need to select a new overall project service pattern.

www.manaraa.com

Compared to Current Methodologies

Nurer et al. (2005) accurately contrast the differences between waterfall and ASD methodologies. They describe the waterfall method as fundamentally assuming that "Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning." On the other hand, ASDs assume "High-quality, adaptive software can be developed by small teams using principles of continuous design improvement and testing based on rapid feedback and change" which allows the initial planning to be performed fast and less costly. Using this basis for comparison, SOSD is best described as a methodology that draws on the strengths of both waterfall and agile methods. Like the waterfall method, a complete project plan and requirements specification is generated upfront in an SOSD project. Like ASDs, these initial steps in the SOSD method require far less time and money than the waterfall method. However, unlike ASDs, these savings are not because only the first 2-4 weeks of the entire project are planned. The efficiency is the result of an increased ability to reuse prior project plans and knowledge. This ability of SOSD is enabled by establishing standardized service descriptions and patterns which are shared and used by all of the PMs in an IS unit.

Practical Issues

While SOSD appears to have significant benefits, it can also create new complexities and interdependencies that must be properly managed so that new problems aren't created. As discussed earlier, as systems become more modular, they also become more complex because of the greater number of elements that must be managed (Baldwin & Clark, 1997). In other words, software development is performed not by a single integrated project team, but a set of loosely-coupled interdependent service providers. As a result, some unexpected problems can occur. For example, if a particular service or service provider is reused too excessively, bottlenecks may be created which can slow down all projects (Lara et al., 2007). PMs must schedule the workflow properly and help to manage the dependencies of the service providers who become bottlenecks. In addition, new interdependencies are created between groups which are not used to working together (e.g. "business side" and "IT side"). They must find ways to collaborate effectively despite cultural differences (Gruman, 2007). Another issue which becomes exacerbated by SOSD is that many of the process interdependencies are between service providers that cross unit and organizational boundaries. If SOSD enables outsourcing to a greater degree, then PMs lose a certain level of control over how the project is performed (Kumaran et al., 2007). They become more vulnerable if the companies they outsource to fail to meet service requirements. Also, SOSD is not mutually exclusive with other currently used software development methodologies (e.g. Waterfall, Spiral, Agile, etc.) (Elssamadisy, 2008). While SOSD changes the way projects are planned, analyzed, and coordinated, traditional methodologies may still be used in the design, development, and implementation phases. For example, SOSD divides the tasks into services and manages the relationships between those services. However, each service provider may use their own unique development methodology to execute the service. One group may use the Waterfall method while another uses SCRUM. All that matters is that the group provides working code. These are just a few of the issues that may arise from SOSD adoption. Each of these problems causes increasing overhead costs to SOFTWARE development projects. While it is difficult to place specific numbers on the sources of these costs, they still add to the overall project cost and completion date.

As described above, each of the mentioned issues with SOSD adoption have the effect of increasing either the complexity or the interdependence of the task environment that the software development unit works within. It should come as no surprise that when SOSD is adopted, organizations may need to make changes to their organization structure, governance structure, and formal and informal coordination practices (Howard, 2008). As with ASD, new methodologies such as SOSD may require cultural changes. If power changes hands when the structure is realigned there may be pushback and resistance.

However, these issues are common with the adoption of many new technologies and methodologies. The primary drawback, or barrier to adoption, with SOSD is that it requires a significant amount of time and effort upfront to build the repository of service patterns and resource allocation patterns from which to search through. Business analysts must take the time to: 1) identify the relevant services (and sub-services, sub-sub-services, etc.) which are needed during typical software projects, 2) define a standardized definition for each service including the required inputs and produced outputs, and 3) create and store an initial set of service patterns, sub-patterns, etc. as well as resource allocation patterns. In addition, standards need to be developed so that the process of creating and storing new patterns as more projects are completed can be accomplished in a way that will allow a shared understanding of the patterns by all PMs. Doing so may also require a greater degree of collaboration between the "IT side" and "business side" than they are used to. The result is that it will take some time after adopting SOSD before the full benefits are realized. Essentially, the repository of "knowledge" must be built up before it is useful. Once this happens though, the results can be impressive. The next section briefly summarizes the implementation results from one software development unit.

www.manaraa.com

SOSD ADOPTION BY ELECTRICAL POWER COMPANY

Over the last 4 years, the authors have helped to implement and/or monitor the results of implementing SOSD in the software development units of a Fortune 100 financial services company as well as a Fortune 500 electrical power company (referred to as "FSC" and "EPC" for non-disclosure purposes). In both cases, SOSD adoption affected similar changes to governance, organization structure, and coordination practices. Also, both organizations were characterized as having a high number of younger professionals still in the early stages of their careers. In addition, both organizations had also undergone structural changes so the employees in each department were fairly new to the others they were working with – not an environment conducive to ASD.

At FSC, we took an action research approach to help implement the new SOSD methodology as requested by the unit director. However, our role at EPC was simply to observe the informal SOSD adoption efforts which were already taking place[2]. They purchased a unique modeling tool which had the ability to create and store process model "objects" which were kept on a centralized server and accessed by each analyst. Over time, they decomposed the high-level models into sub models and categorized them on the server in a manner that allowed each analyst to access each other's objects, sub-models, and overall models.

Table 1 summarizes EPC's SOSD adoption over 11 projects which best capture the range of adoption[3] represented as the both the percent of project patterns which were modeled and the percent of those patterns which were reused from prior projects. Projects one through five had no project patterns modeled at all. However, the new methodology which included modeling the software development process itself and the preparation of reusable models began during project 5. Projects 1-4 represent the characteristics of projects before any SOSD adoption took place while Projects 5 and 6 represent the very early stages of their SOSD adoption. During projects 7, 8 and 9, the new BA group felt they had clearly become more expert in their new methodology and projects 10 and 11 represent when they were able to begin reusing project models (i.e. service patterns) which had been generate during projects 5-9.

Adoption Category	Project	% of patterns reused	% of project patterns modeled[4]
Before Adoption	1	0	0
	2	0	0
	3	0	0
	4	0	0
Early Stage	5	0	0
	6	0	50
Learning Stage	7	0	90
	8	0	70
	9	0	80
Maturing Stage	10	20	100
	11	50	100

Table 1: SOSD Adoption at EPC

The result was that, over time, project cost, schedule, and scope measures became more realistic and the number of scope changes decreased (See Table 2). PMs and the unit director felt that the more detailed modeling that was performed upfront

[2] EPC wasn't adopting a formal SOSD methodology, but rather, their forward-thinking business analysis (BA) unit had the idea of generating reusable process models for both the business process being impacted by each project *and* the project process itself.

[3] Notice that the final project (#11) only reuses 50 percent of its service patterns from prior projects. It is important to note that it would be unlikely that new projects would ever use 100 percent of its service patterns from prior projects because each project will be unique.

[4] The 50 percent of modeled project patterns is the result of the gradual adoption over time. At first, only high level patterns were modeled. With each project, the analysts discovered that they could gain additional benefits from modeling increasingly granular patterns. Hence, the difference in modeling percentages among projects.

المنارة للاستشارات
www.manaraa.com

(with the collaboration of both the IT and business sides), including the greater depth of detail, helped to reduce the uncertainties that cause scope changes down the road. In addition, because the software development process itself had been modeled, the design flaws, technical incompatibilities, etc. which used to show up during the development phase (which caused additional scope changes) were also reduced. As expected, this resulted in a greater amount of cost upfront during the planning and analysis phases. However, the impacts to the overall project cost and schedule were quite impressive as seen in Table 2.

Project	Realistic cost, schedule, and scope estimates?[5]	# of Formal Scope Changes	Changes Normalized by Initial Project Budget	% of budget spent pre-project	% over/ under cost overall	% over/under schedule overall
1	2.1	40	.0000**21**	19.3%	332.9%	95.2%
2	2.6	52	.0000**19**	18.0%	153.0%	99.6%
3	4.1	44	.0000**54**	4.3%	622.7%	205.8%
4	4.7	16	.0000**10**	16.2%	17.1%	424.8%
5	5.1	3	.00000**2**	25.2%	16.2%	34.2%
6	5.8	8	.00000**7**	27.0%	55.1%	29.4%
7	4.6[6]	6	.00000**4**	28.4%	-52.1%	13.9%
8	5.7	7	.00000**1**	27.1%	-14.0%	-27.3%
9	5.8	2	.00000**1**	21.8%	-15.9%	3.9%
10[7]	n/a	n/a	n/a	n/a	n/a	n/a
11[8]	n/a	n/a	n/a	n/a	n/a	n/a

Table 2: Project Performance Metrics over Time[9]

DISCUSSION AND CONCLUSION

Although EPC's ability to meet and exceed project cost and schedule estimates greatly improved as SOSD was adopted, there are likely many other factors which contribute to software project success as well. Therefore, the more telling indicator of the impact of SOSD at EPC may be its ability to reduce uncertainty as indicated by the reduction in scope change. In addition, PMs developed greater confidence in their ability to meet the schedules and budgets as SOSD was increasingly adopted. In summary, EPC was able to significantly reduce project uncertainty, length, and cost by adopting a methodology based on optimizing project planning and analysis upfront rather than "planning for change" or optimizing their ability to handle change throughout the project life cycle. The implication of this finding is not that SOSD is superior to ASD, but rather that it

[5] This measure is derived from interviews performed with the PM of each project, the unit director, and executive sponsor. The interview question is derived from a previously defined Likert-type scale (Kearns & Sabherwal, 2006-7) with 1 = extremely unrealistic and 7 = extremely realistic. The resulting score is the average of the three responses.

[6] Project 7 appears less realistic than those around it chronologically primarily because of some political battles that took place in the early stages. This is because this project required the software development unit being examined here to work with another development unit from a different segment of the organization.

[7] Project 10 is still in progress. Performance data is not yet available.

[8] Project 11 is still in progress. Performance data is not yet available.

[9] Each project involved the development of a new software feature to be added to ECP's customer relationship management system. These projects were selected in particular because they represented the "typical" projects performed by the unit as perceived by the director. In particular, 4 additional projects during the same time period were eliminated from the analysis because they involved tasks other than software development or spanned across atypical project boundaries. The initial budgets for each project ranged from approximately 1 to 3 million and the initial project lengths were estimated to be between 1 and 2 years.

المنارة للاستشارات
www.manaraa.com

can produce some of the same desired changes in environments where ASD may not succeed. Future research should examine any synergies between the two.

Ideally, the impacts of SOSD would be measured across a large sample of organizations of various types. However, this service-oriented concept is still relatively new in the business process context making it difficult to find research participants. But, because the EPC case demonstrates that some organizations are adopting service-oriented principles informally, future research could contribute by identifying informal measures of SOSD principles which could be collected from a wide variety of organizations. Still, because SOSD is somewhat novel, one of the purposes of this research is to provide a methodology and direction for organizations which are seeking the ability to reduce uncertainty and improve agility even if their task environments are not fertile for ASD methodologies. As service-oriented principles continue to be adopted, future research should expand and modify the methodology presented here. In addition, SOSD has its own assumptions and constraints as identified above. Future research should examine these in detail to understand why adopting SOSD may be a bad idea in certain situations. As mentioned before, changes to structure, governance, and coordination may be required. Future research should examine what these changes are from theoretical perspectives such as Organization Theory (Thompson, 1967), Coordination Theory (Malone and Crowston, 1994), Network Theory, and others.

REFERENCES

1. Baldwin, C. T., & Clark, K. B. (1997). Managing in an age of modularity. *Harvard Business Review , 75* (5), 84-93.
2. Bieberstein, N., Bose, S., Walker, L., & Lynch, A. (2005). Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. *IBM Systems Journal , 44* (4), 691-708.
3. Chesbrough, H., & Spohrer, J. (2006). A research manifesto for services science. *Communications of the ACM , 49* (7), 35-40.
4. Dyba, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology , 50*, 833-859.
5. Elssamadisy, A. (2007, April 14). *SOA and Agile: Friends or Foes?* Retrieved February 20, 2009, from InfoQueue: http://www.infoq.com/articles/SOA-Agile-Friends-Or-Foes
6. Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design.* Saddle River, NJ: Prentice Hall.
7. Foster, I. (2005). Service-Oriented Science. *Science , 308*, 814-817.
8. Gruman, G. (2007, May 14). *Breaking SOA Bottlenecks*. Retrieved February 20, 2009, from InfoWorld: http://www.infoworld.com/article/07/05/14/20FEsoabottle-intro_1.html
9. Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly , 28* (1), 75-105.
10. Highsmith, J., & Cockburn, A. (2001). Agile Software Development: The Business of Innovation. *Computer , 34* (9), 120-122.
11. Hoetker, G. (2006). Do modular products lead to modular organizations? *Strategic Management Journal , 27* (6), 501-518.
12. Howard, C. (2008, March 17). *Tips to Avoid SOA Implementation Problems.* Retrieved February 20, 2009, from http://www.cio.com: http://www.cio.com/article/197956/Tips_to_Avoid_SOA_Implementation_Problems
13. Kumaran, S., Bishop, P., Chao, T., Dhoolia, P., Jain, P., & Jaluka, R. (2007). Using a model-driven transformational approach and service-oriented architecture for service delivery management. *IBM Systems Journal , 46* (3), 513-529.
14. Langlois, R. N. (2002). Modularity in technology and organization. *Journal of Economic Behavior & Organization , 49* (1), 19-37.
15. Lara, R., Corella, M. A., & Castells, P. A. (2007). Flexible model for locating services on the web. *International Journal of Electronic Commerce , 12* (2), 11-40.
16. MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., & Metz, R. (2006, October 12). *OASIS Reference Model for Service Oriented Architecture 1.0.* Retrieved February 20, 2009, from www.oasis-open.org: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

www.manaraa.com

17. Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005). Challenges of Migrating to Agile Methodologies. *Communications of the ACM, 48*(5), 73-78.
18. Newcomer, E. (2003). *Understanding web services: XML, WSDL, SOAP, and UDDI.* Boston, MA: Addison-Wesley.
19. Oracle. (2008, August). *State of the Business Process Management Market 2008.* Retrieved February 20, 2009, from www.oracle.com: http://www.oracle.com/technologies/bpm/docs/state-of-bpm-market-whitepaper.pdf
20. Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM , 15* (12), 1053-1058.
21. Raschke, R., & David, J. S. (2005). Business Process Agility. *Proceedings of the 11th Americas Conference on Information Systems*, (pp. 355-360). Omaha, NE.
22. Schmidt, R., Lyytinen, K., Keil, M., & Cule, P. (2001). Identifying Software Project Risks: An International Delphi Study. *Journal of Management Information Systems , 17* (4), 5-36.
23. Sheth, A., Verma, K., & Gomadam, K. (2006). Semantics to energize the full services spectrum. *Communications of the ACM , 49* (7), 55-61.
24. Sutherland, J., Viktorov, A., Blount, J., & Puntikov, N. (2007). Distributed Scrum: Agile Project Management with Outsourced Development Teams. *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)* (p. 274a). Waikoloa, HI: IEEE.
25. Thompson, J. D. (1967). *Organizations in Action.* New York: McGraw-Hill.
26. Turk, D., France, R., & Rumpe, B. (2005). Assumptions Underlying Agle Software-Development Processes. *Journal of Database Management , 16* (4), 62-87.

المنارة للاستشارات
www.manaraa.com