

Web Services Implementation Methodology for SOA Application

Siew Poh Lee
Singapore Institute of Manufacturing Technology
71, Nanyang Drive
638075
SINGAPORE
spllee@SIMTech.a-star.edu.sg

Lai Peng Chan
Singapore Institute of Manufacturing Technology
71, Nanyang Drive
638075
SINGAPORE
lpchan@SIMTech.a-star.edu.sg

Eng Wah Lee
Singapore Institute of Manufacturing Technology
71, Nanyang Drive
638075
SINGAPORE
ewlee@SIMTech.a-star.edu.sg

Abstract – With the popularity of Web Services technology and the increase trend of developing Service-Oriented Architecture (SOA) software, there is a need for an implementation methodology for Web Services. This paper addresses the SOA application development challenges, identifies gaps in agile software methodology for Web Services development, and observes Web Services characteristics and its best practices. The contribution of this paper is to extend any existing agile software methodology to include Web Services Best Practices into the agile software methodology.

I. INTRODUCTION

The Service-Oriented Architecture (SOA) is a software architecture that defines the use of services, to support software user requirements [2]. The characteristics of these services are reusable business components; loosely coupled; building blocks of SOA application with the intent to provide services to either end user applications or other services through published and heterogeneous network addressable software component.

The implementation of SOA application is made possible through the realization of Web Services. Web Service is a software component representing specific set of business functions that can be described, published and invoked over the Internet using XML-based open standards such as SOAP [3], WSDL [4] and UDDI [5]. The SOA application development involves developing software components for software reuse and wrapping software components as Web Services for end user applications or other services consumptions. However, there are gaps in the existing software component development methodology as it does not include the design and development factors specific for Web Services.

This paper discusses the research work done in bridging the gaps by investigating the characteristics and the best practices of Web Services development. The outcome of the research work is part of the specification developed for OASIS Framework for Web Services Implementation (FWSI) [13] Implementation Methodology Sub-Committee [15] (IMSC) in the effort to develop implementation methodology for Web Services.

II SOA APPLICATION DEVELOPMENT CHALLENGES

In a dynamic digital economy, the demand for direct process integration to business partners from organisations and sharing of information is increasing. Organisations are gradually turning to SOA application to increase their business agility and IT productivity. The building blocks for SOA application come from different service providers. The SOA application development process is complex and

tedious. It requires proper software project management planning and control to ensure systematic approach for handling and developing SOA application. This is attributed to the fact that some of these building blocks of services are outside company's boundary. The confronted challenges for SOA application development are:

1. Difficulty in capturing of user requirements as the requirements no longer derives from a single source. It can come from multiple stakeholders who may be geographically distributed.
2. Difficulty in collating different services as not all the services are implemented using the same technology. Some services could be implemented using C++, C#, Java, PERL, etc. The hosting of services on different technology platforms also contributed to the collating difficulties.
3. Difficulty in services consumption due to different types of services provided. Some services only support asynchronous interaction; some may support synchronous interaction, etc. This poses difficulty in SOA application.
4. Difficulty in services communication because of different interfaces, e.g. document-oriented message exchange; parameters-oriented message exchange, supported by different services.
5. Different services offer different degrees of service coupling. Document-based services are more loosely couple than non document-based (i.e. parameter-based) services.
6. Complicated task in conducting SOA application test as it requires a well-coordinated effort from all services' providers to ensure all services are available.

To address the challenges mentioned, we need to analyze the characteristics of Web Services and the best practices for Web Services development in order to explore the possibility of enhancing the existing software development methodology with the inclusion of Web Services specific activities for SOA application development. With this as our research basis, our approach for carrying this research is described in the following section.

III RESEARCH APPROACH

Our approach is to first study an existing agile methodology and identifies gaps in the methodology. The second is to study the Web Services development steps to identify the steps specific to Web Services. The third is to study the Web Services characteristics and its best practices.

The result of these is the proposed methodology for Web Services development. The workflow of our approach is shown in Figure 1.

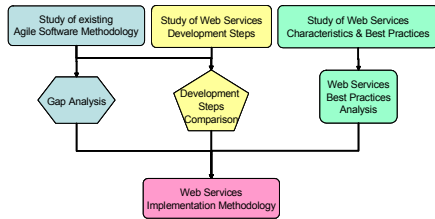


Fig. 1 Research Approach

IV ANALYSIS STUDY

A. Software Components, Web Services and SOA Application

A *component* is a reusable software building block: a pre-built piece of encapsulated application code that can be combined with other components and with additional code to produce custom application [1]. A *software component* is an independently deliverable package of reusable software services. Software components are the reusable building blocks of SOA application. The relationship of software component, Web Services and SOA application is shown in Figure 2. The development of Web Services is based on software component through public interfaces exposed for services consumption. For example, *Order Analyzer* and *Order Generator* are software components derived from objects/classes. The *Order Processor* is a web service that uses components *Order Analyzer* and *Order Generator* to provide richer business functionality as building blocks for SOA application.

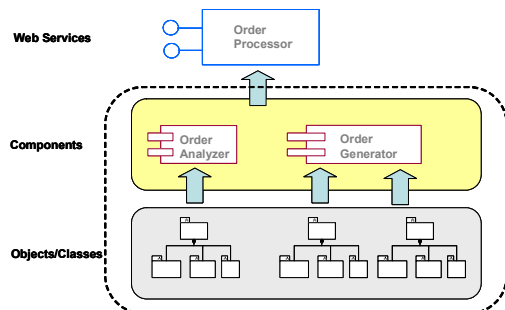


Fig. 2. Web Services CBD Development

B. Agile Software Development

Component-based development is a software development approach where the entire lifecycle of the software creation, development deployment and maintenance is centred on the *start-to-finish* concept of component lifecycle. Figure 3 depicts a component-based development (CBD) process. The CBD process has five phases, namely *Requirements*, *Analysis*, *Design*, *Implementation* and *Testing*. Artifact is produced at each phase which in turn is the inputs to different types of testing shown in Figure 3. The component has its own lifecycle and

it is related with the whole system lifecycle. Agile software development can be applied in component-based software development. Further reading for the adaptation of agile techniques into traditional software methodology for Rational Unified Process (RUP), for example, is described in [19]. Any of the agile software development methodologies such as extreme programming (XP) [11], IBM Rational Unified Process (RUP) [11], etc can be applied to component-based software development:

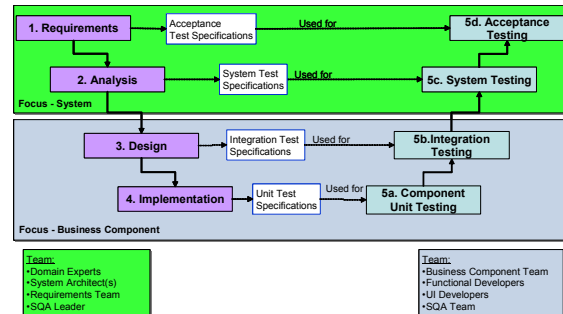
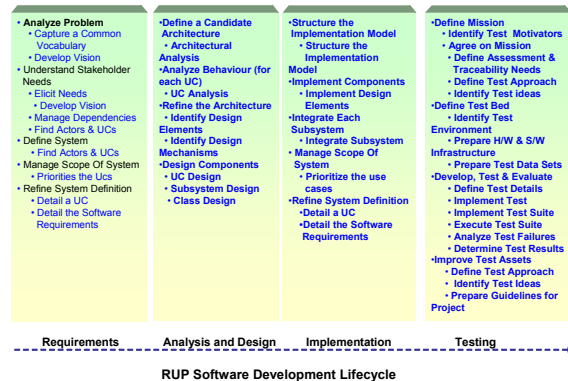


Fig. 3. An Illustration of CBD Process

The following Figure 4 gives a snapshot of software development lifecycle activities, for RUP.



RUP Software Development Lifecycle

Fig. 4. A typical software development lifecycle activities

The context for Web Services activities in each of the software lifecycle phases is absent from the methodology. The best approach of bridging this gap is to analyze the special characteristics of Web Services and its best practices to be considered at different phases of the software lifecycle. The following section discusses the Web Services development steps and its characteristics.

C. Web Services Development

Figure 5 shows the workflow for Web Services development. There are eight steps, namely:

1. Gather user requirements.
2. Analyze business components to be reused or create new service.
3. Design the Web Service (WS).
4. Develop WS by implementing business logic with the used of *interface* and *implementation* classes.

The interface class is where the service interface will be exposed for consumption and the implementation class is actual implementation of the services derived from software components

5. Build WS by wrapping component into WS.
6. Deploy WS to the target web server based on the deployment script (which is server specific).
7. Test and debug WS using web service client (where the client is server specific).
8. Publish WS if publishing to service registry is required.

As shown in Figure 5, Step 5: Build, Step 6: Deploy and Step 8: Publish are specific to Web Services development as compared to component-based development. Step 7: WS Test requires a platform specific WS client to test the WS. The artifacts generated from these steps are also specific to Web Services. The outputs (i.e. interface and implementation classes) produced from Step 4 are specific to Web Services as well.

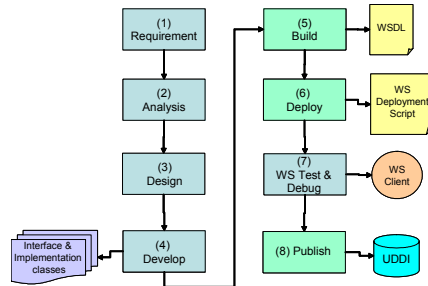


Fig. 5. Web Services Development Workflows

By comparing agile software development steps and Web Services development steps, we are able to extend Web Services specific steps into the agile development methodology

D. Web Services Characteristics and Best Practices

The realization of SOA is centered on Web Services (WS). It is important to understand fully the characteristics of Web Services, in terms of the dos and don'ts for WS, form the basis of the best practices for Web Services development. These characteristics affect the design and implementation of Web Services. The following sub-sections discuss the characteristics of Web Services and its associated best practices.

WSBP1. Web Services Styles. There are two most common styles of Web Services, namely remote procedure call (RPC) style WS and document style WS. The differences between these two styles are summarized in Table 1. The RPC-styled offers simplicity and better tooling support. The document-styled offers greater flexibility and decoupling of services.

TABLE 1 Web Services Styles

| | RPC-Styled | Document-Styled |
|------------------------|------------------------------------|--------------------------------|
| <i>Processing Mode</i> | Business object-centric | Document-centric |
| <i>Interaction</i> | Request and Response (Synchronous) | Fire and forget (Asynchronous) |
| <i>Implementation</i> | Java, EJB | JMS |

WSBP2. Web Services Interaction Mode. Web Services have four interaction modes. They are *synchronous* interaction (i.e. request and wait for response), *asynchronous* interaction (i.e. fire and forget), *solicit-response* interaction (i.e. the service sends a message followed by a correlated message from client), and *notification* interaction (i.e. the service sends a message). Any one of this mode will affect the way of designing and implementation Web Services.

WSBP3. Web Services Client Implementation – Interaction Mode. The client implementation will be determined by Web Services Interaction modes. If it is an asynchronous WS, an asynchronous WS client implemented using Java API for XML Messaging JAXM, for example, will be used. Otherwise, Java API for XML for Remote Procedure Call (JAX-RPC) will be used.

WSBP4. Web Services Client Implementation – Client Types. The client implementation is affected by the types of Web Services client. Particularly in Java-based RPC service, there are three different types of Web Services client, namely static stub, dynamic proxy and dynamic invocation proxy (DII) of web service clients for consuming a service. The three types of client offer different degree of client flexibility. For example, static stub is the least flexible as any changes to the service would require rebuilding of service client. DII is the most flexible as the client parses the service's WSDL in constructing a SOAP message for service invocation. Any change to the end point service does not require rebuilding of client.

WSBP5. Right Level of Service Interface Granularity. The granularity of the service interface affects the design and implementation of web service. In addition, it also affects the performance of the service. The finer the granularity for service interface, the slower the performance as it is an overhead to the network and drop in web service performance.

WSBP6. Interoperability. Interoperability issues could be caused by different versions of SOAP standard implementation, different types of security algorithms for digital signature, encryption/decryption, and variation in supporting Web Services standards from multiple vendors. The adoption of primitive data type for parameters whenever possible. Avoid using customized SOAP serializer/deserializer and different types of encoding standards.

WSBP7. **Binding Style** The use of *rpc/encoded* or *doc/literal* binding style is determined by the needs of data information being exchange between Web Service client and the service. If it is data intensive or the exchanged information is a file, then *doc/literal* binding is preferred. If the data information exchanged is relatively static, then *rpc/encoded* binding is preferred.

WSBP8. **Request and Response Performance.** Web Services itself is network intensive. It demands extra network bandwidth and CPU processing time and memory due to the needs for SOAP message serialization and de-serialization overhead. The common practices for optimizing the request and response performance are: (a) perform data caching in either client or server side (b) decide Web Services operation granularity, (c) Use XML judiciously in document-centric Web Services by careful considering whether to use whole or segment of XML document.

WSBP9. **Security.** There are various ways to secure the information sent between initial SOAP sender and ultimate SOAP receiver via numerous intermediary SOAP nodes. Different means of security can affect the way how Web Services are designed and implemented. The security means can be through:

- a. *Transport Level Security* (TLS). In this mean, it leverages on transport security mechanism. Only the initial SOAP sender and ultimate SOAP receiver are secured. Intermediary nodes are not secured. The two most common means are secure socket layer (SSL) or HTTPS.
- b. *Message Level Security* (MLS). In this mean, message can be secured throughout the whole SOAP message path. Standards such as XML Encryption⁰, XML Signature⁰, XML Key Management⁰, WS-Security⁰, SAML^[9], etc. can be applied to secure the XML message and
- c. *Infrastructural Level Security*. In this mean, it leverages on the security mechanism provided by Web Services hosting platform.

WSBP10. **Web Services Implementation Technology and Platform.** What is the technology platform, such as J2EE or .NET based, to be used? What kind of application server is required to host services? The understanding these lead to better services interoperability.

WSBP11. **Industry Standard Conformance.** The conformance of industry standard, such as RosettaNetTM⁰ and ebXML⁰ provided by the service determines the type of services. As it gives rise to the consideration of the requirement for well-formed XML document and document-styled Web Service for the service.

WSBP12. **Addressable Software Component.** Every end point service is identifiable using universal resource locator (URL). To know whether service is available, an invocation test to the service URL would provide the availability status of the service.

WSBP13. **Web Services Needs.** Web Services Technology is applied to meet certain business needs and objectives. The considering factors include reuse business components, integrate different IT platforms and disparate islands of technologies, direct business-to-business integration (B2Bi) between partners to facilitate information sharing. Understanding the basic needs would ascertain better drive of Web Services Technology to be applied appropriately.

WSBP14. **Web Services Layering Architecture.** The consideration for hierarchical abstraction for Web Services enables the decoupling relationship for services. This facilitates layered hierarchy representing ordered grouping of functionality abstraction for domain-specific application (upper layer), across domains application (middle layer), and deployment environment-specific application (lower layer).

The best practices for Web Services (i.e. the dos and don'ts of Web Services) will be based on the characteristics of Web Services listed. Understanding the best practices of Web Services helps us in addressing the SOA design and implementation challenges discussed earlier.

V. OUR METHODOLOGY

The analysis of the gaps of software methodology for Web Services (WS) and study of Web Services characteristics and best practices discussed earlier complement each other. This forms our basis of extending the existing agile software methodology with Web Services best practices (WSBP).

Our major effort is to go through every activities and tasks defined for each phase of the software lifecycle by analyzing how the Web Services best practices could fit in. The phases identified to be suitable for Web Service implementation lifecycle are: requirements, analysis, design, implementation, test, and deployment. The following sub-sections describe the consideration of WSBP discussed earlier

A. Requirements Phase

The objective of this phase is to understand the business requirements and translating them into WS requirement in terms of the features, the functional and non-functional requirements, and the WS constraint. The WSBP13 provides a guideline for identifying Web Services, categorizing the needs into Web Services. The features required for the respective Web Services. Define use case models for the respective Web Services.

B. Analysis Phases

The objective of this phase is to refine the requirements further and translate the requirements into conceptual models. Architecting analysis is done to define high-level structure and identify the Web Services interfaces contracts. The WSBP1, WSBP2, WSBP5, WSBP6, WSBP10 and WSBP11 provide analysis guidelines for the following activities:

- Analyzing the granularity of Web Services interface contracts.
- Selecting technology platform for implementation framework.
- Defining Web Services candidate architecture. Identify architectural components to be exposed as WSs and specify major information exchanged with client.

C. Design Phase

The objective of this phase is to do detail design of Web Service. In this phase, the WS interface is refined further. The interaction of between the service and the client, e.g. asynchronous/synchronous or rpc/document is considered. The Web Services best practices for WSBP1, WSBP2, WSBP3, WSBP5, WSBP6, WSBP7 and WSBP14 are considered.

D. Implementation Phase

The objective of this phase is to do the actual coding of Web Services. The wrapping of components APIs to Web Services interface is done. The generations of WSDL and WS test client are produced. The WS will be deployed to the target application server. The WS best practices for WSBP1 to WSBP11 provide the guidelines for the implementation of Web Services.

E. Testing Phase

The objective of this phase is to conduct a complete test for Web Services including functional and non-functional requirements. The WS best practices for WSBP1 to WSBP11 provide guidelines for the testing of Web Services.

F. Deployment Phase

The objective of this phase is to ensure Web Service is properly deployed to the targeted application server. To validate proper deployment of WS, the server specific WS client is used to conduct the deployment.

From WSBP1, the user would specify if the publishing of their web services is required for internal organisation, or extended to their trading partners or external used. This leads to the decision whether to have a private or public service registry to serve their company's needs. If there is a need to publish to a service registry, then activity for gathering additional information for registry publishing is

considered for the phase.

Figure 5 depicts the overview of the extended software methodology lifecycle that has incorporation Web Services best practices in different phases of the lifecycle.

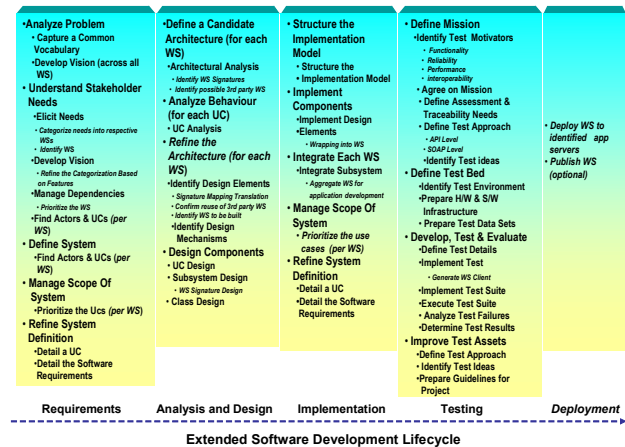


Fig. 6 Extended Methodology

VI. FWSI WEB SERVICES IMPLEMENTATION METHODOLOGY

The outcome of this research study is the key contribution to the OASIS FWSI TC Implementation Methodology Sub-Committee (IMSC) for Web Services Implementation Methodology (WSIM) guidelines [13].

The purpose of FWSI TC is to facilitate implementation of robust Web Services by defining a *practical and extensible methodology* consisting of *implementation processes* and common functional elements that practitioners can adopt to create high quality Web Services systems without re-inventing them for each implementation by defining only the *Web Services-specifics activities* spans across software development lifecycle.

The methodology itself is iterative and incremental. The Web Service would go through all the phases thereby developing and refining the Web Services throughout the project lifecycle per iteration. As compared with the normal structured methodology and agile software methodology, the WSIM has an extra *deployment* phase after the *Testing* phase. This phase is specific to Web Services as the developed services need to be deployed and hosted in a targeted application server that provides the reference implementation for Web Services Architecture [14] defined by W3C. Figure 6 outlines the FWSI WSIM Web Services specific activities.

VII. CONCLUSION

Service-oriented architecture application faces a number of challenges. The nature of SOA application is centred on software components. Web Services technology facilitates the realization of SOA application. The investigation of existing agile software methodology for component-based development is analyzed for identifying the gaps for Web Services development. The comparison study between Web

Services development and the agile methodology are conducted to identify the additional steps required for Web Services development. In addition, the study of Web Services characteristics and the best practices are analyzed. The practical approach is proposed to extend the existing agile methodology with the inclusion of Web Services best practices to every phases of agile software lifecycle. Without reinventing a new software methodology, a systematic and practical methodology for SOA application development is developed. Our research team has done a detailed case study on applying the methodology mentioned to IBM-Rational RUP and extreme programming (XP). These two case study reports [20], [22] are available at OASIS FWSI IMSC. The reports show the web services specific activities and tasks for RUP and XP software development phases respectively. The research team is currently conducting survey with the selected Singapore companies ranging from end users to IT solution providers, to validate against the WSIM based on the analysis made WSIM.

Springer-Verlag London 2006, pp 193 – 204.

- [20] OASIS FWSI IMSC Web Services Implementation Methodology – Rational Unified Process (Example) Working Draft 02, published 23rd December, 2004
- [21] OASIS FWSI IMSC Web Services Implementation Methodology – Case Example using Extreme Programming, published 14th August 2005

VIII. REFERENCES

- [1] S.H. Kaisler *Software Paradigms*, John Wiley & Son, 2005, pp 99 – 100
- [2] H. Hao “What is Service-Oriented Architecture” xml.com published 30th Sep. 2003.
- [3] SOAP Version 1.2 Part 0 Primer W3C Recommendation published 24th June 2003
- [4] WSDL Version 1.1 W3C Note published 15th March 2001
- [5] Evolution of UDDI The Stencil Group White Paper published 19th July 2002 at UDDI.org
- [6] XML Encryption Syntax and Processing Specification W3C Recommendation published 10th December 2002
- [7] XML Signature Syntax and Processing W3C Recommendation published 12th February 2002
- [8] XML Key Management Specification v2.0 W3C Recommendation published 28th June 2005
- [9] Technical Overview of the OASIS Security Assertion Markup Language (SAML) Version 1.1 Committee Draft published 11th May 2004
- [10] Web Services Security: SOAP Message Security 1.1 OASIS Standard Specification published 1st February 2006
- [11] Agile Software Construction John Hunt Springer
- [12] IBM RUP Training Course Materials
- [13] OASIS Framework for Web Services Implementation Technical Committee FWSI
- [14] Web Services Architecture W3C Working Group Note published 11th February 2004
- [15] OASIS FWSI IMSC version 1.0 guidelines
- [16] Deciphering RosettaNet™ WebMethods Whitepaper published April 2000.
- [17] ebXML Technical Architecture Specification version 1.04 published 16th February 2001
- [18] A.S. Koch, *Agile Software Development Evaluating the Methods for Your Organisation* Artech House, 2005
- [19] John Hunt, *Agile Software Construction*,