



Web Service Implementation Methodology

Public Review Draft 1.0, 05 September 2005

Document identifier:

fwsim-1.0-guidelines-doc-wd-PublicReviewDraft1.0.pdf

Location:

http://www.oasis-open.org/committees/documents.php?wg_abbrev=fwsi

Editors:

Eng Wah LEE, Singapore Institute of Manufacturing Technology
<ewlee@SIMTech.a-star.edu.sg>
Marc HAINES, individual <mhaines@uwm.edu>

Contributors:

Lai Peng CHAN, Singapore Institute of Manufacturing Technology
<lpchan@SIMTech.a-star.edu.sg>
Chai Hong ANG, Singapore Institute of Manufacturing Technology
<chang@SIMTech.a-star.edu.sg>
Puay Siew TAN, Singapore Institute of Manufacturing Technology
<pstan@SIMTech.a-star.edu.sg>
Han Boon LEE, Singapore Institute of Manufacturing Technology
<hblee@SIMTech.a-star.edu.sg>
Yushi CHENG, Singapore Institute of Manufacturing Technology
<ycheng@SIMTech.a-star.edu.sg>
Xingjian XU, Singapore Institute of Manufacturing Technology
<xjxu@SIMTech.a-star.edu.sg>
Zunliang YIN, Singapore Institute of Manufacturing Technology
<zlyin@SIMTech.a-star.edu.sg>

Abstract:

This document specifies Web Service specific activities in a Web Service Implementation Methodology and illustrates the approach to incorporate these activities into an existing agile software development methodology.

Status:

Committee members should send comments on this specification to the fwsim-imsclists@lists.oasis-open.org list. Others should subscribe to and send comments to the fwsicomment@lists.oasis-open.org list. To subscribe, send an email message to fwsicomment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the FWSI TC web page (<http://www.oasis-open.org/committees/fwsi/>).

Table of Contents

45	1	Introduction	8
46	1.1	Purpose	8
47	1.2	Target Audience	8
48	1.3	Scope	8
49	1.3.1	Not in Scope	8
50	2	Implementation Methodology Overview	10
51	2.1	Objective	10
52	2.2	Web Service Implementation Lifecycle	10
53	2.3	Phase	10
54	2.3.1	Requirements Phase	11
55	2.3.2	Analysis Phase	11
56	2.3.3	Design Phase	12
57	2.3.4	Coding Phase	12
58	2.3.5	Test Phase	12
59	2.3.6	Deployment Phase	12
60	2.4	Role	13
61	2.5	Glossary	13
62	3	Web Service Implementation Methodology	14
63	3.1	Overview	15
64	3.2	Requirements Phase	16
65	3.2.1	Activity: Determine the need for Web Service	16
66	3.2.1.1	Tasks	16
67	3.2.1.2	Roles	16
68	3.2.1.3	Artifacts	16
69	3.2.2	Activity: Elicit Web Service requirements	16
70	3.2.2.1	Tasks	16
71	3.2.2.2	Roles	16
72	3.2.2.3	Artifacts	16
73	3.2.3	Activity: Manage the Web Service requirements	17
74	3.2.3.1	Tasks	17
75	3.2.3.2	Roles	17
76	3.2.3.3	Artifacts	17
77	3.2.4	Activity: Model the usage scenarios	17
78	3.2.4.1	Tasks	17
79	3.2.4.2	Roles	17
80	3.2.4.3	Artifacts	17
81	3.2.5	Activity: Prepare Test Cases for User Acceptance Test (UAT) and System Test	17
82	3.2.5.1	Tasks	17
83	3.2.5.2	Roles	18
84	3.2.5.3	Artifacts	18
85	3.3	Analysis Phase	18
86	3.3.1	Activity: Select a technology platform as implementation framework	18
87	3.3.1.1	Tasks	18
88	3.3.1.2	Roles	19
89	3.3.1.3	Artifacts	19
90	3.3.2	Activity: Define a candidate architecture for the Web Service	19

91	3.3.2.1 Tasks	19
92	3.3.2.2 Roles	19
93	3.3.2.3 Artifacts.....	19
94	3.3.3 Activity: Decide on the granularity of the Web Service.....	19
95	3.3.3.1 Tasks	19
96	3.3.3.2 Roles	19
97	3.3.3.3 Artifacts.....	20
98	3.3.4 Activity: Identify reusable Web Services.....	20
99	3.3.4.1 Tasks	20
100	3.3.4.2 Roles	20
101	3.3.4.3 Artifacts.....	20
102	3.3.5 Activity: Identify service interface for new Web Services	20
103	3.3.5.1 Tasks	20
104	3.3.5.2 Roles	20
105	3.3.5.3 Artifacts.....	20
106	3.3.6 Activity: Prepare Test Cases for Performance Test	20
107	3.3.6.1 Task.....	20
108	3.3.6.2 Roles	21
109	3.3.6.3 Artifacts.....	21
110	3.3.7 Activity: Prepare Test Cases for Integration / Interoperability Test	21
111	3.3.7.1 Task.....	21
112	3.3.7.2 Roles	21
113	3.3.7.3 Artifacts.....	21
114	3.3.8 Activity: Prepare Test Cases for Functional Test	21
115	3.3.8.1 Task.....	21
116	3.3.8.2 Roles	21
117	3.3.8.3 Artifacts.....	21
118	3.3.9 Activity: Testbed preparation	21
119	3.3.9.1 Task.....	21
120	3.3.9.2 Roles	21
121	3.3.9.3 Artifacts.....	21
122	3.4 Design Phase	23
123	3.4.1 Activity: Transform signatures of reusable Web Services	23
124	3.4.1.1 Tasks	23
125	3.4.1.2 Roles	23
126	3.4.1.3 Artifacts.....	23
127	3.4.2 Activity: Refine service interface of the new Web Service.....	23
128	3.4.2.1 Tasks	23
129	3.4.2.2 Roles	23
130	3.4.2.3 Artifacts.....	23
131	3.4.3 Activity: Design Web Service	23
132	3.4.3.1 Tasks	23
133	3.4.3.2 Roles	24
134	3.4.3.3 Artifacts.....	24
135	3.4.4 Activity: Refine Test Cases for Functional Test.....	24
136	3.4.4.1 Task.....	24
137	3.4.4.2 Roles	24
138	3.4.4.3 Artifacts.....	24
139	3.5 Coding Phase.....	24
140	3.5.1 Activity: Construct Web Service code.....	24
141	3.5.1.1 Tasks	24

142	3.5.1.2 Roles	25
143	3.5.1.3 Artifacts.....	25
144	3.5.2 Activity: Construct Web Service client code	25
145	3.5.2.1 Tasks	25
146	3.5.2.2 Roles	25
147	3.5.2.3 Artifacts.....	25
148	3.5.3 Activity: Unit Test Web Service.....	26
149	3.5.3.1 Tasks	26
150	3.5.3.2 Roles	26
151	3.5.3.3 Artifacts.....	26
152	3.6 Test Phase	26
153	3.6.1 Activity: Test functionality of Web Service	27
154	3.6.1.1 Tasks	27
155	3.6.1.2 Roles	27
156	3.6.1.3 Artifacts.....	27
157	3.6.2 Activity: Integration Test on the Web Service.....	27
158	3.6.2.1 Tasks	27
159	3.6.2.2 Roles	28
160	3.6.2.3 Artifacts.....	28
161	3.6.3 Activity: System Test on the Web Service	28
162	3.6.3.1 Tasks	28
163	3.6.3.2 Roles	28
164	3.6.3.3 Artifacts.....	28
165	3.6.4 Activity: User Acceptance Test on the Web Service	28
166	3.6.4.1 Tasks	28
167	3.6.4.2 Roles	28
168	3.6.4.3 Artifacts.....	28
169	3.7 Deployment Phase	29
170	3.7.1 Activity: Prepare deployment environment	29
171	3.7.1.1 Tasks	29
172	3.7.1.2 Roles	29
173	3.7.1.3 Artifacts.....	29
174	3.7.2 Activity: Deploy Web Service	29
175	3.7.2.1 Tasks	29
176	3.7.2.2 Roles	30
177	3.7.2.3 Artifacts.....	30
178	3.7.3 Activity: Test deployment.....	30
179	3.7.3.1 Tasks	30
180	3.7.3.2 Roles	30
181	3.7.3.3 Artifacts.....	30
182	3.7.4 Activity: Create end user support material.....	30
183	3.7.4.1 Tasks	30
184	3.7.4.2 Roles	30
185	3.7.4.3 Artifacts.....	30
186	3.7.5 Activity: Publish Web Service	31
187	3.7.5.1 Tasks	31
188	3.7.5.2 Roles	31
189	3.7.5.3 Artifacts.....	31
190	4 References.....	32
191	Appendix A. Acknowledgments	33
192	Appendix B. Revision History	34

193 Appendix C. Notices 35
194

List of Figures

196 Figure 1: Web Service Implementation Lifecycle 11
197 Figure 2: The “V” Model incorporates the Web Services specific Interoperability test..... 15
198 Figure 3: Relationship between phase, activities, tasks, roles and artifacts 15
199

List of Tables

201	Table 1: Mapping between phases and roles assigned	12
202	Table 2: Overview of activities, tasks, roles and artifacts in the Requirements Phase	18
203	Table 3: Overview of activities, tasks, roles and artifacts in the Analysis Phase	22
204	Table 4: Overview of activities, tasks, roles and artifacts in the Design Phase	24
205	Table 5: Overview of activities, tasks, roles and artifacts in the Coding Phase	26
206	Table 6: Overview of activities, tasks, roles and artifacts in the Test Phase.....	29
207	Table 7: Overview of activities, tasks, roles and artifacts in the Deployment Phase	31
208		
209		

210 1 Introduction

211 1.1 Purpose

212 The purpose of this document is to define a practical and extensible Web Service
213 Implementation Methodology that can be used as a reference for Web Services development
214 and deployment. This document is a consolidation of the best practices by Web Services
215 practitioners and aims to improve the Web Services implementation process through the
216 formalization of a Web Service implementation lifecycle and defining Web Service specific
217 activities and artifacts.

218
219 This document should be used in conjunction with the Functional Elements¹ specifications to
220 govern the approach by which the Functional Elements are implemented.
221

222 1.2 Target Audience

223 The target audiences are likely to be:

- 224
- 225 • Project Managers
226 This document provides a formal methodology for Web Services implementation, which
227 can be used for management and control.
 - 228 • Software Architects/Designers/Developers/Testers
229 This document identifies activities that are repeatable and which can be abide by, so as to
230 ensure the quality of the software produced.
231

232 1.3 Scope

233 This document focuses Web Service specific activities, artifacts, roles and responsibilities that
234 can be incorporated into an existing agile software development methodology (e.g. RUP,
235 Extreme Programming, Feature Driven Development etc). For a few common agile
236 methodologies the technical committee is preparing examples that show in detail how the
237 generic activities, artifacts, roles, and responsibilities described in this document can be
238 incorporated and used in a given methodology. These case examples are provided in
239 separate documents that will be published along with this document when they become
240 available. Currently the technical committee is preparing cases for RUP and Extreme
241 Programming (XP).

242
243

244 1.3.1 Not in Scope

245 This document does not define yet another novel software development methodology.
246 Instead, the Web Service implementation methodology highlights important features in the
247 context of Web Services. The elements of the Web Service implementation methodology are
248 based on existing agile software methodology and extend it by incorporating Web Service
249 specific activities.

250
251 Also, it is not in the scope of this document to specifically address how each of these software
252 development methodologies should be tailored to incorporate Web Service specific parts.
253 Examples are provided only to illustrate just one possible way of tailoring a specific agile
254 development methodology for Web Service implementation.
255

¹ The Functional Elements are to be specified as components, which are to be exposed as Web Services where appropriate.

256 This document does not intend to define a new software development methodology. Instead,
257 the Web Service Implementation Methodology leverages on an existing agile software
258 methodology and extend it by incorporating the Web Services specific activities.
259
260 This document also does not cover the detailed description or explanation of any of the
261 existing agile software development methodology nor does it recommend one particular agile
262 software development methodology over another.

263 2 Implementation Methodology Overview

264 2.1 Objective

265 The Web Service Implementation Methodology defines a systematic approach to Web
266 Service development by leveraging on an agile software development methodology and
267 extending that methodology by specifying the Web Services specific activities and the
268 corresponding roles and work-products that are produced in the process.

269
270 This methodology will define a set of common practices that create a method-independent
271 framework, which can be applied by most software teams for developing Web Service
272 applications.

273
274

275 2.2 Web Service Implementation Lifecycle

276 A *Web Service Implementation Lifecycle* refers to the phases for developing Web Services
277 from requirement to deployment.

278

279 The Web Service implementation lifecycle typically includes the following phases:

- 280 1. Requirements Phase [see 2.3.1]
- 281 2. Analysis Phase [see 2.3.2]
- 282 3. Design Phase [see 2.3.3]
- 283 4. Coding Phase [see 2.3.4]
- 284 5. Test Phase [see 2.3.5]
- 285 6. Deployment Phase [see 2.3.6]

286

287 The transitions through these phases need not be a single-pass sequential process. On the
288 contrary, the process tends to be iterative and incremental in nature and should be agile
289 enough to accommodate revisions in situations where the scope cannot be completely
290 defined up front.

291

292

293 2.3 Phase

294 A *Phase* when used in the context of a Web Service implementation lifecycle refers to the
295 period of time a set of related software implementation activities are carried out.

296

297 In general, the phases detailed in the sub-sections are identified to be pertinent in a Web
298 Service implementation lifecycle. These phases may overlap with each other in the course of
299 the implementation process as shown in Figure 1.

300

301

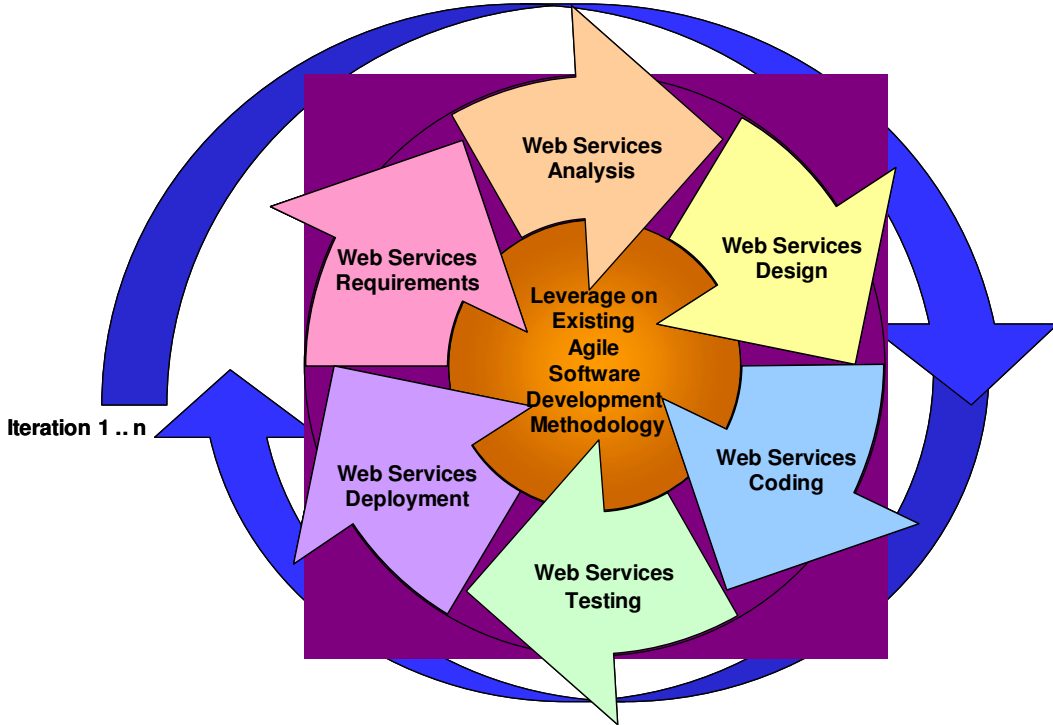


Figure 1: Web Service Implementation Lifecycle

303
304
305
306
307

2.3.1 Requirements Phase

The objective in the requirements phase is to understand the business requirements and translating them to Web Service requirements in terms of the features, the functional and non-functional requirements, and the constraints within which the Web Service has to abide.

Requirements elicitation should be done by the requirements analyst and should involve the project stakeholders such as the project champion, customers, end users, etc. Following which, the analyst should interpret, consolidate and communicate these requirements to the development team.

If possible, Requirements should be aggregated in a centralized repository where they can be viewed, prioritized, and “mined” for iterative features. In all cases, enabling the team to easily capture requirements, search, prioritize and elaborate as necessary is the primary function of the repository.

2.3.2 Analysis Phase

In the analysis phase, the requirements of the Web Service are further refined and translated into conceptual models by which the technical development team can understand. It is also in this phase that an architecture analysis is done to define the high-level structure and identify the Web Service interface contracts. This process should be performed by both the requirements analyst and the architect and communicated to the design and development teams.

323
324
325
326
327
328
329
330

331 **2.3.3 Design Phase**

332 The detailed design of Web Services is done in this phase. In this phase, the designers
333 should define the Web Service interface contract that has been identified in the analysis
334 phase. The defined Web Service interface contract should identify the elements and the
335 corresponding data types (possibly using a XML schema) as well as mode of interaction
336 between the Web Service and the client, for example, whether it should be
337 synchronous/asynchronous or RPC/Document style etc.
338

339 **2.3.4 Coding Phase**

340 The coding and debugging phase for Web Service implementation is essentially quite similar
341 to other software component-based coding and debugging phase. The main difference lies in
342 the creation of additional Web Service interface wrappers (to expose the components' public
343 APIs), generation of WSDLs and client stubs. Web Services in addition have to be deployed
344 to a Web Server/Application Server before the test clients can consume them.
345

346 The component developer and/or the tester should perform these activities.
347

348 **2.3.5 Test Phase**

349 For testing of Web Services, besides testing for functional correctness and completeness,
350 testers should also perform interoperability testing between different platforms and clients'
351 programs. Furthermore, performance testing has to be conducted to ensure that the Web
352 Services are able to withstand the maximum load and stress as specified in the non-functional
353 requirements specification. Other tasks like profiling of the Web Service application and
354 inspection of SOAP messages should also be done in this phase.
355

356 **2.3.6 Deployment Phase**

357 The purpose of the deployment phase is to ensure that the Web Service is properly deployed.
358 The phase will be executed after the service has been tested. The deployment of the Web
359 Service is platform specific. The service end points of the Web Service specifies where the
360 service is deployed and it needs to be identified and configured accordingly. The deployer
361 primary tasks are to ensure that the Web Service has been properly configured and managed
362 (e.g. version controlled, presetting of configuration files, packaged and loaded in the correct
363 location etc.) and running post-deployment tests to ensure that the Web Service is indeed
364 ready for use. Other optional tasks like specifying and registering the Web Service with an
365 UDDI registry may also be performed in this phase.
366

367 Table 1 summaries the overview of each phase against its' respective assigned roles.
368

Phases	Primary Roles
Requirements	Requirements Analysts
Analysis	Requirements Analysts Architects
Design	Designers
Coding	Developers Testers
Test	Testers
Deployment	Deployers

369

370

Table 1: Mapping between phases and roles assigned

371

372 **2.4 Role**

373 Commonly defined roles in software development methodology include the following:
 374

Roles	Responsibilities
Requirements Analyst	Responsible for eliciting and interpreting the stakeholders' needs, and communicating those needs to the entire team.
Architect	Responsible for the software architecture, which includes the key technical decisions that constrain the overall design and implementation for the project.
Designer	Responsible for designing a part of the system, within the constraints of the requirements, architecture, and development process for the project.
Developer	Responsible for developing and unit testing the components, in accordance with the project's adopted standards.
Deployer	Responsible for planning the product's transition to the user community, ensuring those plans are enacted appropriately, managing issues and monitoring progress.
Stakeholder	Responsible for providing the domain expertise and specifying the system requirements. Stakeholder usually includes the project champion and the end users.
Project Manager	Responsible for managing and monitoring the project including the project scope, schedule and staffing of the project team.
Test Manager	Responsible for the total test efforts including the quality and test advocacy, resource planning and management of the testing schedule, and resolution of issues that impede the test effort.
Test Designer	Responsible for defining the test approach and ensuring its successful implementation. The role involves identifying the appropriate techniques, tools and guidelines to implement the required tests, and to give guidance on the corresponding resources requirements for the test effort. The role also involves monitoring detailed testing progress and results in each test cycle and evaluating the overall quality as a result of testing activities.
Tester	Responsible for the core activities of the test effort, which involves conducting the necessary tests and logging the outcomes of that testing.
System Administrator	Responsible for planning, installing and maintaining the hardware and software of the different environments e.g. development, test, live environment

375
 376

377 **2.5 Glossary**

378

Activity	An Activity refers to a unit of work a role may be assigned to perform. Activities are performed within each of the phases in the Web Service implementation lifecycle.
Artifact	An <i>Artifact</i> refers to the work-product that is used or produced as a result of performing an activity. Examples of Artifacts include models, source files, scripts, and binary executable files.
Role	A <i>Role</i> refers to the responsibilities that a person or a team has been assigned with.

379

380

3 Web Service Implementation Methodology

381

The term *Web Service* describes a specialized type of software, which is designed to support a standardized way for provision and consumption of services over the Web, through the compliance with open standards such as eXtensible Markup Language (XML), SOAP, Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI).

382

383

384

385

386

387

Web Services, unlike traditional client/server systems, such as browser/Web server systems, are not meant for direct end-user consumption. Rather, Web Services are pieces of business logic, which have programmatic interfaces and it is through these interfaces that developers can create new application systems.

388

389

390

391

392

The motivation behind Web Services is to facilitate businesses to interact and integrate with other businesses and clients, without having to go through lengthy integration design and/or to expose its confidential internal application details unnecessarily. This is made possible by leveraging on the non-platform dependent and non-programming language dependent XML to describe the data to be exchanged between businesses or between the business and its clients, using a WSDL to specify what the service is providing; using a UDDI to publish and locate who is providing the service; and typically using SOAP over HTTP to transfer the message across the internet².

393

394

395

396

397

398

399

400

401

A Web Service, naturally, is a software element, but because of its specialized interface and mechanism to interoperate with others, all the prevalent generic software development methodology would need to be tailored to handle the unique features of Web Service. This could translate to identification of Web Service specific requirements (e.g. conformance to Web Services standards), analysis of the specific implications of Web Service on the overall system, design of the Web Service interface and XML message structure, coding, testing, deployment and execution of the Web Service.

402

403

404

405

406

407

408

409

The Web Service Implementation Methodology that we define is to promote a systematic approach to Web Service development. Rather than defining a new software development methodology and expecting software practitioners to forget their own familiar and established methodology to re-learn another, the better alternative is to leverage on what is already available and customize that methodology to incorporate the specifics of Web Services.

410

411

412

413

414

415

The candidate software development methodology should, ideally, be agile and able to accommodate refinement throughout the development cycle in an iterative and incremental approach. The methodology should consist of phases that cover from the conception of the need of the Web Service, to the construction of the Web Service and finally to be deployed for use by the eventual client application. In this document, these phases are identified as requirements, analysis, design, code, test and deployment.

416

417

418

419

420

421

422

The Web Service Implementation Methodology would leverage on any of the candidate agile software development methodology and extend the said methodology by specifying additional and/or customized Web Service specific activities and its corresponding roles and work-products.

423

424

425

426

427

The Web Service Implementation Methodology is iterative and incremental. In each iteration, the Web Service would go through all the phases (i.e. requirements, analysis, design, code, testing and finally deployment), thereby developing and refining the Web Services throughout the project lifecycle.

428

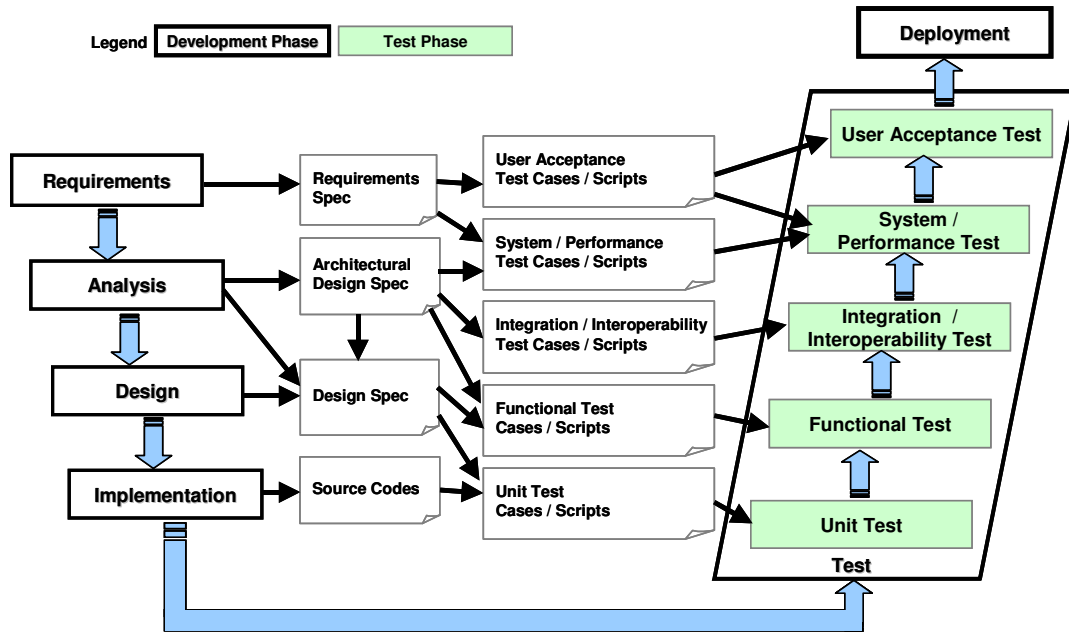
429

430

431

² SOAP is transport agnostic. Therefore, other Internet (e.g. SMTP) or non-Internet (IBM MQ Series) may be used. From a practical perspective, however, SOAP over HTTP appears to be the typical scenario.

432 In addition, for Web Service testing, a multitude of tests have to be conducted to ensure that
 433 the Web Service is developed according to its functional as well as non-functional
 434 requirements. Figure 2 illustrates using the “V” Model to perform these tests.
 435



436

437

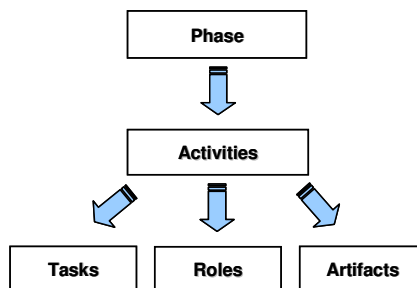
438 *Figure 2: The “V” Model incorporates the Web Services specific Interoperability test*

439

440 The specifications produced in each of the development phases are sources of input to derive
 441 the test scenarios and test cases. From these test cases, test scripts and test data are
 442 compiled, which will be used in unit testing, functional testing, integration/interoperability
 443 testing, system/performance testing and the final user acceptance testing.
 444

445 3.1 Overview

446 The Web Service Implementation Lifecycle describes the phases a typical Web Service would
 447 undergo, from the identification of the need of the Web Service to the final deployment and
 448 usage by the end-users. The phases identified to be relevant in the Web Service
 449 Implementation Lifecycle are: requirements, analysis, design, code, test and deployment. In
 450 each of these phases, Web Service specific activities are carried out. These activities, as well
 451 as the roles and responsibilities, and the artifacts will be elaborated in the subsequent sub-
 452 sections. Figure 3 illustrates the above-mentioned relationship between phase, activities and
 453 their respective tasks, roles and artifacts.
 454



455

456 *Figure 3: Relationship between phase, activities, tasks, roles and artifacts*

457 **3.2 Requirements Phase**

458 **3.2.1 Activity: Determine the need for Web Service**

459 **3.2.1.1 Tasks**

- 460 • Identify the stakeholders
461 Stakeholders would usually include the end users, project champion, project
462 manager, etc.
463
- 464 • Understand the inadequacies/problems to address
465 Understand the stakeholders' need for Web Services.
466
- 467 • Identify the need for Web Service technology
468 Based on the current technology available, identify needs specially for Web Services.
469
- 470 • Determine the positioning of the Web Service within the boundaries of the problem
471 identified
472
- 473 • Define the features of the Web Service based on the needs list
474
- 475 • Identify the limitations to be imposed on the Web Service

476 **3.2.1.2 Roles**

477 Architect, Requirements Analyst, Stakeholders, Project Manager

478 **3.2.1.3 Artifacts**

479 The results should be recorded in Business Requirement Specifications.
480

481 **3.2.2 Activity: Elicit Web Service requirements**

482 **3.2.2.1 Tasks**

- 483 • Identify the sources for requirements gathering based on the features list
484 Identify the departments, end users, domain experts, etc. who would be impacted by
485 the introduction of Web Services.
486
- 487 • Gather information from these sources and elicit the requirements for the Web
488 Service
489
- 490 • Identify functional requirements for the Web Service and categorise them
491
- 492 • Identify non-functional requirements for the Web Service
493 Non-functional requirements are requirements pertaining to Usability, Reliability,
494 Performance, Scalability, Supportability and other design considerations.

495 **3.2.2.2 Roles**

496 Requirements Analyst, Architect, Test Manager

497 **3.2.2.3 Artifacts**

498 The results should be recorded in Requirement Specifications.
499

500 **3.2.3 Activity: Manage the Web Service requirements**

501 **3.2.3.1 Tasks**

- 502 • Based on the functional requirements categories, identify the Web Services and
503 establish the dependencies and priorities
504
- 505 • Create traceability matrices from the requirements to the identified Web Services
506 Traceability matrices help to track the requirements that have been taken care of by
507 the Web Services identified.
508
- 509 • Manage changes to the requirements

510 **3.2.3.2 Roles**

511 Requirements Analyst, Architect, Test Manager

512 **3.2.3.3 Artifacts**

513 The results should be recorded in Requirement Specifications.
514

515 **3.2.4 Activity: Model the usage scenarios**

516 **3.2.4.1 Tasks**

- 517 • Translate the functional requirements into conceptual usage models using some form
518 of analysis modeling techniques
519
- 520 • Specify the major interaction scenarios with the Web Service clients
521 This is to highlight the usage of Web Services involved. Especially, the message
522 exchange scenarios should be captured.

523 **3.2.4.2 Roles**

524 Requirements Analyst, Architect, Test Manager

525 **3.2.4.3 Artifacts**

526 The results should be recorded in Requirement Specifications.
527

528 **3.2.5 Activity: Prepare Test Cases for User Acceptance Test (UAT)** 529 **and System Test**

530 **3.2.5.1 Tasks**

- 531 • Write business scenario test case(s) based on the requirements gathered to be used
532 for UAT and System Test
533 Test case(s) can be derived from requirements. This is also a way to verify the
534 requirements when they are implemented.
535
- 536 • Build requirement validation matrix
537 The requirement validation matrix will include the requirements and a reference to the
538 test case(s) that will validate the requirement.
539
- 540 • Manage changes to the test cases when requirements changed

541 **3.2.5.2 Roles**

542 Requirements Analyst, Test Manager, Test Designer

543 **3.2.5.3 Artifacts**

544 The results should be recorded in Test Plan – UAT and System Test.

545

546 Table 2 summaries the overview of each activities and the corresponding tasks, roles and
547 artifacts under the activities.

548

Activities	Tasks	Roles	Artifacts
<ul style="list-style-type: none"> ▪ Determine needs 	<ul style="list-style-type: none"> ▪ Identify stakeholders ▪ Understand the inadequacies/problems to address ▪ Identify need for WS technology ▪ Determine positioning of WS within the boundaries of the problem identified ▪ Define features of WS based on needs ▪ Identify limitations 	<ul style="list-style-type: none"> ▪ Architect ▪ Requirements Analyst ▪ Stakeholders ▪ Project Manager 	<ul style="list-style-type: none"> ▪ Business Requirement Specifications
<ul style="list-style-type: none"> ▪ Elicit requirements 	<ul style="list-style-type: none"> ▪ Identify sources for requirements gathering ▪ Gather information ▪ Identify functional requirements ▪ Identify non-functional requirements 	<ul style="list-style-type: none"> ▪ Architect ▪ Requirements Analyst ▪ Test Manager 	<ul style="list-style-type: none"> ▪ Requirement Specifications
<ul style="list-style-type: none"> ▪ Manage requirements 	<ul style="list-style-type: none"> ▪ Identify WS and establish dependencies and priorities ▪ Create traceability matrices ▪ Manage changes to requirements 	<ul style="list-style-type: none"> ▪ Architect ▪ Requirements Analyst ▪ Test Manager 	<ul style="list-style-type: none"> ▪ Requirement Specifications
<ul style="list-style-type: none"> ▪ Model usage scenarios 	<ul style="list-style-type: none"> ▪ Translate functional requirements into conceptual usage models ▪ Specify major interaction scenarios with WS clients 	<ul style="list-style-type: none"> ▪ Architect ▪ Requirements Analyst ▪ Test Manager 	<ul style="list-style-type: none"> ▪ Requirement Specifications
<ul style="list-style-type: none"> ▪ Prepare test cases for UAT and System Test 	<ul style="list-style-type: none"> ▪ Write business scenarios test cases ▪ Build requirement validation matrix ▪ Manage changes to test cases 	<ul style="list-style-type: none"> ▪ Requirements Analyst ▪ Test Manager ▪ Test Designer 	<ul style="list-style-type: none"> ▪ Test Plan – UAT and System Test

549

550 Notes: WS stands for Web Services

551

552 *Table 2: Overview of activities, tasks, roles and artifacts in the Requirements Phase*

553

554

555 **3.3 Analysis Phase**

556 **3.3.1 Activity: Select a technology platform as implementation**
557 **framework**

558 **3.3.1.1 Tasks**

- 559 • Specify the Web Services standards that the implementation must adhere
- 560 Identify the Web Service standards based on the requirements and implementation
- 561 constraints. Consider issues like the standards compatibility, version of standards,
- 562 standards adoption in industry sector, and the organization approving the standards.
- 563
- 564 • Decide the technology platform for implementing Web Services
- 565 Choose a technology platform that is suitable for implementation. E.g. dotNet or Java
- 566 platform.
- 567
- 568 • Decide the technology platform for hosting Web Services

569 Based on implementation constrains and considerations for standards support and
570 interoperability requirements, choose the appropriate hosting platform for Web
571 Services.
572
573 • Decide the IDE tools used to develop Web Services
574 Available options include commercial vendor's IDE tools, open source IDE tools.
575 Normally, the selection of IDE is tied together with the implementation platforms.

576 **3.3.1.2 Roles**

577 Architect

578 **3.3.1.3 Artifacts**

579 The results should be recorded in Software Architecture Specifications.
580

581 **3.3.2 Activity: Define a candidate architecture for the Web Service**

582 **3.3.2.1 Tasks**

- 583 • Define a high-level architecture
584
- 585 • Identify the architectural component that expose functionality as Web Services
586 It is necessary to identify the architectural components that implement the wrapping
587 of functionality as Web Services and implement the message exchanges in the high
588 level architecture.
- 589 • Specify the major information exchange with Web Service clients
590 Identify and specify the first cut definition of message that is exchanged with Web
591 Services clients. The definition includes the element of data, data type and format.
592

593 **3.3.2.2 Roles**

594 Architect

595 **3.3.2.3 Artifacts**

596 The results should be recorded in Software Architecture Specifications.
597

598 **3.3.3 Activity: Decide on the granularity of the Web Service**

599 **3.3.3.1 Tasks**

- 600 • Decide on the coarseness of the Web Service operations to be exposed
601 Set up criteria on the coarseness of Web Services operations. Its definition depends
602 on the usage scenarios and requirements.
603
- 604 • Identify and group functionality into the Web Service
605 Based on the requirements and criteria mentioned above, identify the functions that
606 are needed to group into the Web Services.
607
- 608 • Decide on the mechanisms to compose or aggregate functionality
609 In case there is a need to compose individual Web Services, choose and decide the
610 mechanism to implement the compositions.

611 **3.3.3.2 Roles**

612 Architect

613 **3.3.3.3 Artifacts**

614 The results should be recorded in Software Architecture Specifications.
615

616 **3.3.4 Activity: Identify reusable Web Services**

617 **3.3.4.1 Tasks**

- 618 • Identify the architectural components that can be realized by existing Web Services
619 If the functionality of the architecture component can be fulfilled with existing Web
620 Services (internal or third party Web Services), the architectural components should
621 be identified to make use of these existing Web Services.
622
- 623 • Identify the Web Service providers for the reusable Web Services
624 Identify and gather the information about provider of existing Web Services.
625
- 626 • Define the major invocation scenarios of re-use
627 Identify the functions that are going to be used. Define the interface of invocation.

628 **3.3.4.2 Roles**

629 Architect

630 **3.3.4.3 Artifacts**

631 The results should be recorded in Software Architecture Specifications.
632

633 **3.3.5 Activity: Identify service interface for new Web Services**

634 **3.3.5.1 Tasks**

- 635 • Define the new Web Service operation signatures
636 Based on the usage models and analysis models, identify the operations and its
637 signatures.
638
- 639 • Define XML schema for the message exchange
640 If message exchanges are involved, the XML schema that guides the structure of the
641 message should be defined.

642 **3.3.5.2 Roles**

643 Architect, Designer

644 **3.3.5.3 Artifacts**

645 Web Service Signature Specifications, XML schema.
646

647 **3.3.6 Activity: Prepare Test Cases for Performance Test**

648 **3.3.6.1 Task**

- 649 • Write performance test case(s) to be used for Performance Test
650 Test case(s) can be derived from Architectural Design Specifications.
651
- 652 • These test cases should cover load testing scenarios to see how the system will
653 perform under various loads (in terms of concurrent users/requests and/or
654 transactions).

655 **3.3.6.2 Roles**

656 Test System Administrator, Test Designer

657 **3.3.6.3 Artifacts**

658 The results should be recorded in Test Plan – Performance Test.
659

660 **3.3.7 Activity: Prepare Test Cases for Integration / Interoperability**
661 **Test**

662 **3.3.7.1 Task**

- 663 • Write integration / interoperability test case(s) to be used for Integration /
664 Interoperability Test
665 Test case(s) can be derived from Architectural Design Specifications.

666 **3.3.7.2 Roles**

667 Test Designer, Tester

668 **3.3.7.3 Artifacts**

669 The results should be recorded in Test Plan – Integration / Interoperability Test.
670

671 **3.3.8 Activity: Prepare Test Cases for Functional Test**

672 **3.3.8.1 Task**

- 673 • Write functional test case(s) to be used for Functional Test
674 Test case(s) can be derived from Architectural Design Specifications.

675 **3.3.8.2 Roles**

676 Test Designer, Tester

677 **3.3.8.3 Artifacts**

678 The results should be recorded in Test Plan - Functional Test.
679

680 **3.3.9 Activity: Testbed preparation**

681 **3.3.9.1 Task**

- 682 • Set up testing environment that include hardware and software
683
684 • This environment may be similar to the production/live environment in terms of
685 hardware, OS, Web Server/Application Server, etc.

686 **3.3.9.2 Roles**

687 Test System Administrator, Test Designer

688 **3.3.9.3 Artifacts**

689 The results should be recorded in Test Plan - Testbed.
690

691
692
693

Table 3 summaries the overview of each activities and the corresponding tasks, roles and artifacts under the activities.

Activities	Tasks	Roles	Artifacts
<ul style="list-style-type: none"> Select a technology platform as implementation framework 	<ul style="list-style-type: none"> Specify implementation standards Decide technology platform for implementation Decide technology platform for hosting Decide IDE tools used for development 	<ul style="list-style-type: none"> Architect 	<ul style="list-style-type: none"> Software Architecture Specifications
<ul style="list-style-type: none"> Define candidate architecture 	<ul style="list-style-type: none"> Define high-level architecture Identify architectural component that expose functionality as WS Specify major information exchange with WS clients 	<ul style="list-style-type: none"> Architect 	<ul style="list-style-type: none"> Software Architecture Specifications
<ul style="list-style-type: none"> Decide granularity 	<ul style="list-style-type: none"> Decide on coarseness of the operations to be exposed Identify and group functionality Decide on mechanisms to compose or aggregate functionality 	<ul style="list-style-type: none"> Architect 	<ul style="list-style-type: none"> Software Architecture Specifications
<ul style="list-style-type: none"> Identify reusable WS 	<ul style="list-style-type: none"> Identify architectural components that can be realized by existing WS Identify WS providers for reusable WS Define major invocation scenarios of re-use 	<ul style="list-style-type: none"> Architect 	<ul style="list-style-type: none"> Software Architecture Specifications
<ul style="list-style-type: none"> Identify service interface 	<ul style="list-style-type: none"> Define new WS operation signatures Define XML schema for message exchange 	<ul style="list-style-type: none"> Architect Designer 	<ul style="list-style-type: none"> WS Signature Specifications XML Schema
<ul style="list-style-type: none"> Prepare test cases for Performance Test 	<ul style="list-style-type: none"> Write Performance test cases 	<ul style="list-style-type: none"> Test System Administrator Test Designer 	<ul style="list-style-type: none"> Test Plan – Performance Test
<ul style="list-style-type: none"> Prepare test cases for Integration / Interoperability Test 	<ul style="list-style-type: none"> Write Integration / Interoperability test cases 	<ul style="list-style-type: none"> Test Designer Tester 	<ul style="list-style-type: none"> Test Plan – Integration / Interoperability Test
<ul style="list-style-type: none"> Prepare test cases for Functional Test 	<ul style="list-style-type: none"> Write functional test cases 	<ul style="list-style-type: none"> Test Designer Tester 	<ul style="list-style-type: none"> Test Plan – Functional Test
<ul style="list-style-type: none"> Testbed preparation 	<ul style="list-style-type: none"> Set up testing environment 	<ul style="list-style-type: none"> Test System Administrator Test Designer 	<ul style="list-style-type: none"> Test Plan - Testbed

694
695
696
697
698
699

Notes: WS stands for Web Services

Table 3: Overview of activities, tasks, roles and artifacts in the Analysis Phase

700 **3.4 Design Phase**

701 **3.4.1 Activity: Transform signatures of reusable Web Services**

702 **3.4.1.1 Tasks**

- 703 • Identify the data type mapping if required
704 If the type of a parameter of the reusable service is not directly supported by the
705 identified platform, data type mapping should be performed.
706
- 707 • Identify the design patterns for mapping the re-used Web Service interface to the
708 identified (desired) one
709 Certain design patterns could be used to reuse existing Web Service(s), such as
710 adapter pattern, façade pattern etc. Adapter pattern could be used to expose a new
711 interface of an existing Web Service. The façade pattern could be used to
712 encapsulate the complexity of existing Web Services and provide a coarse-grained
713 Web Service.

714 **3.4.1.2 Roles**

715 Designer

716 **3.4.1.3 Artifacts**

717 The results should be recorded in Design Specifications.

718

719 **3.4.2 Activity: Refine service interface of the new Web Service**

720 **3.4.2.1 Tasks**

- 721 • Refine Web Service interfaces signature
722 In the detailed design stage, the signature may be refined further. Care must be
723 taken to ensure that the design decision should not affect the interoperability of the
724 service.
725
- 726 • Refine XML schema for message exchange
727 The XML schema may be refined to further expand on the data structure, data types,
728 namespaces etc.

729 **3.4.2.2 Roles**

730 Designer

731 **3.4.2.3 Artifacts**

732 The results should be recorded in Design Specifications.

733

734 **3.4.3 Activity: Design Web Service**

735 **3.4.3.1 Tasks**

- 736 • Use some form of modeling techniques to describe the internal structure of the Web
737 Service
738 The design of the internal structure needs to consider the receiving and pre-
739 processing of request, delegating of the request, processing of the request and
740 sending of the response. Existing modeling techniques such as UML, design
741 patterns could be applied to the design.
742

- 743 • Consider non-functional requirements (e.g. usability, reliability, performance,
744 scalability etc.) and design constraints (e.g. interoperability etc.)

745 **3.4.3.2 Roles**

746 Designer

747 **3.4.3.3 Artifacts**

748 The results should be recorded in Design Specifications.
749

750 **3.4.4 Activity: Refine Test Cases for Functional Test**

751 **3.4.4.1 Task**

- 752 • Refine functional test case(s) to be used for functional Test
753 Test case(s) can be refined by Design Specifications.

754 **3.4.4.2 Roles**

755 Test Designer, Tester

756 **3.4.4.3 Artifacts**

757 The results should be recorded in Test Plan – Functional Test.
758
759

760 Table 4 summaries the overview of each activities and the corresponding tasks, roles and
761 artifacts under the activities.
762

Activities	Tasks	Roles	Artifacts
<ul style="list-style-type: none"> ▪ Transform signatures of reusable WS 	<ul style="list-style-type: none"> ▪ Identify data type mapping if required ▪ Identify design patterns for mapping the re-used WS interface to the desired one 	<ul style="list-style-type: none"> ▪ Designer 	<ul style="list-style-type: none"> ▪ Design Specifications
<ul style="list-style-type: none"> ▪ Refine service interface of new WS 	<ul style="list-style-type: none"> ▪ Refine WS interface signature ▪ Refine XML schema for message exchange 	<ul style="list-style-type: none"> ▪ Designer 	<ul style="list-style-type: none"> ▪ Design Specifications
<ul style="list-style-type: none"> ▪ Design WS 	<ul style="list-style-type: none"> ▪ Use modeling techniques to describe internal structure of WS ▪ Consider non-functional requirements and design constraints 	<ul style="list-style-type: none"> ▪ Designer 	<ul style="list-style-type: none"> ▪ Design Specifications
<ul style="list-style-type: none"> ▪ Refine test cases for Functional Test 	<ul style="list-style-type: none"> ▪ Refine functional test cases 	<ul style="list-style-type: none"> ▪ Test Designer ▪ Tester 	<ul style="list-style-type: none"> ▪ Test Plan – Functional Test

763 Notes: WS stands for Web Services
764
765

766 *Table 4: Overview of activities, tasks, roles and artifacts in the Design Phase*
767
768

769 **3.5 Coding Phase**

770 **3.5.1 Activity: Construct Web Service code**

771 **3.5.1.1 Tasks**

- 772 • Based on the implementation language choice, code the Web Service according to
773 the design

- 774 Consider other constraints that are imposed by the specific implementation language
775 itself. For example, consider the language dependent data types and the need to
776 map these data types to the ones specified by the Web Service interface.
777
778 • Expose public APIs as Web Service interface
779 For example, in Java, to create the interface class to expose the class method as a
780 Web Service operation or in dotNet, to annotate the class API as a [WebMethod].
781
782 • Generate WSDL for client to consume
783 Most IDEs can auto-generate the WSDL from the interface code.

784 **3.5.1.2 Roles**

785 Developer

786 **3.5.1.3 Artifacts**

787 Web Service Implementation Codes.
788

789 **3.5.2 Activity: Construct Web Service client code**

790 **3.5.2.1 Tasks**

- 791 • Decide on the Web Service Client programming model
792 Among the three available are:
793
794 a) Static Stub
795 The client invokes the Web Service operation through a stub. Any IDE can generate
796 this stub at compile time.
797
798 b) Dynamic Proxy
799 As the name implies, dynamic proxy is dynamically generated when the client
800 application is executed. Because dynamic proxy is generated during runtime, Web
801 Service invocation using this method takes the longest time amongst the three
802 approaches.
803
804 c) DII (Dynamic Invocation Interface)
805 It is the most flexible approach among the three programming models. The client
806 does not even need to know the signature of the Web Service operation until runtime.
807 The Web Service invocation can be dynamically constructed.
808
809 Hence, identify and decide on a suitable client programming model based on the
810 weightage of flexibility against performance requirements.
811
812 • Write client code to consume the Web Service
813 Use the WSDL to generate client stubs, which can be used in the client code to
814 invoke the methods provided by the Web Service.

815 **3.5.2.2 Roles**

816 Developer

817 **3.5.2.3 Artifacts**

818 Web Service Client Codes.
819

820 **3.5.3 Activity: Unit Test Web Service**

821 **3.5.3.1 Tasks**

- 822 • Deploy Web Service in local test environment and perform functional unit testing
- 823 The emphasis is on the correctness of the functionality and the exceptions handling.

824 **3.5.3.2 Roles**

825 Developer

826 **3.5.3.3 Artifacts**

827 Unit Test Scripts.

828

829

830 Table 5 summaries the overview of each activities and the corresponding tasks, roles and
831 artifacts under the activities.

832

Activities	Tasks	Roles	Artifacts
<ul style="list-style-type: none"> Construct WS code 	<ul style="list-style-type: none"> Code based on implementation language chosen Expose public APIs as interface Generate WSDL for client 	<ul style="list-style-type: none"> Developer 	<ul style="list-style-type: none"> Implementation codes
<ul style="list-style-type: none"> Construct WS client code 	<ul style="list-style-type: none"> Decide client programming model Write client codes 	<ul style="list-style-type: none"> Developer 	<ul style="list-style-type: none"> Client codes
<ul style="list-style-type: none"> Unit test 	<ul style="list-style-type: none"> Deploy in local test environment and perform functional unit testing 	<ul style="list-style-type: none"> Developer 	<ul style="list-style-type: none"> Unit test scripts

833

834

Notes: WS stands for Web Services

835

836

Table 5: Overview of activities, tasks, roles and artifacts in the Coding Phase

837

838

839 **3.6 Test Phase**

840 For Web Services, additional tests may be conducted to ensure that the Web Services are
841 interoperable, secured and scalable.

842

843 Interoperability is an issue in Web Services because the standards governing Web Services
844 are still evolving. Furthermore, different vendors that implement these specifications may
845 interpret and comply with these specifications differently. Currently there is an effort by Web
846 Services Interoperability Organization (WS-I) to recommend basic profiles to minimise these
847 incompatibilities. The aim of conducting interoperability tests is to ensure that these
848 recommendations are followed and the Web Service developed will interoperate with other
849 Web Services and products without problems.

850

851 Network congestion created by Web Services is the major contributor to Web Services' slow
852 performance. Not only is the messaging between requesters and Web Services impacted by
853 network latency, but also the service discovery and description protocols that precede those
854 message exchanges. The cumulative effect of these delays can seriously degrade the
855 performance of Web Services. Therefore it is necessary to do a performance test on the Web
856 Services before they are deployed for operation, and then to monitor the Web Services to
857 determine if they can meet the service level agreements.

858

859 Web Services introduce special security issues e.g. in privacy, message integrity,
860 authentication and authorization. Tests have to be conducted to ensure that these security
861 requirements have been fulfilled. However, security schemes could complicate the process of
862 testing and debugging Web Service basic functionality. For example, non-intrusive monitors

863 are often used in functional testing but encrypted traffic presents an obvious complication to
864 this approach to testing.
865

866 **3.6.1 Activity: Test functionality of Web Service**

867 **3.6.1.1 Tasks**

- 868 • Testing basic Web Service functionality
869 The Web Service should respond correctly to requests from their clients. The format
870 of the SOAP message should be in compliance with the specifications. WSDL files,
871 which contain metadata about Web Services' interfaces, should be in compliance with
872 the WSDL specifications published by W3C. Perform fault checking to see how it
873 handles unexpected input. The test scripts and data prepared in the earlier phases
874 are executed in this activity. The test results should be recorded, and bugs found
875 should be reported to the code owners and fixed by them.
876
- 877 • Test for security
878 If a service requires a certain level of privacy, or if it requires that messages be
879 authenticated in a certain way, then specific tests are needed to ensure that these
880 security requirements are met. The test scripts and test data prepared in the earlier
881 phases should be executed in this activity. Any inadequacies that may lead to
882 possible security breaches should be reported and resolved by the code owner,
883 designer or architect.
884
- 885 • Test the UDDI functionality
886 If a service is registered to a registry server, perform registering of Web Service, then
887 write test clients to perform finding and binding of Web Service on the registry, and
888 then use the registry data to actually invoke the service. Test results from the test
889 scripts and data should be recorded and bugs should be fixed by the code owners.
890
- 891 • Test for SOAP intermediary capability
892 If particular SOAP message has one or more intermediaries along the message route
893 that take actions based upon the instructions provided to them in the header of the
894 SOAP message. Web Service SOAP intermediary testing must verify the proper
895 functionality of these intermediaries. Test results from the test scripts and data
896 should be recorded and bugs should be fixed by the code owners.

897 **3.6.1.2 Roles**

898 Tester, Test Designer

899 **3.6.1.3 Artifacts**

900 The results should be recorded in Client Test Code, Test Scripts and Test Results.
901

902 **3.6.2 Activity: Integration Test on the Web Service**

903 **3.6.2.1 Tasks**

- 904 • Test for conformance to Web Services Interoperability Organization (WS-I)
905 recommendations. Execute test scripts and data according to the test cases based
906 on the WS-I recommendations.
907
- 908 • Perform interoperability testing based on various scenarios
909 This is to highlight the interoperability issues of Web Services implementation. Refer
910 to Interoperability Guideline for the interoperability testing scenarios.

911

912

- Perform integration testing based on various scenarios

913

Based on the test cases prepared in the Analysis Phase, test scripts and test data, which are prepared are executed and analyzed in this activity.

914

915 **3.6.2.2 Roles**

916 Tester, Test Designer, Test System Administrator

917 **3.6.2.3 Artifacts**

918 The results should be recorded in Client Test Code, Test Scripts and Test Results.

919

920 **3.6.3 Activity: System Test on the Web Service**

921 **3.6.3.1 Tasks**

- Check system functionality and response time under different degrees of load increases

923

The test cases that are prepared in the earlier phases are executed in this activity.

924

The load increases can be sudden surges or gradual ramp-ups. The test results

925

should be analyzed to determine potential bottlenecks and if the system is scalable.

926

927

- Check functionality and response time under different combinations of valid and invalid requests

928

The results from the test execution should be analyzed to determine if the system can

929

still render the expected quality of service as specified in the non-functional

930

requirement specifications.

931

932

933 **3.6.3.2 Roles**

934 Tester, Test Designer, Test System Administrator

935 **3.6.3.3 Artifacts**

936 The results should be recorded in Client Test Code, Test Scripts and Test Results.

937

938 **3.6.4 Activity: User Acceptance Test on the Web Service**

939 **3.6.4.1 Tasks**

- Run the user acceptance test cases(s) for the Web Services system
The test cases prepared in the Requirement Phase are used in this activity to validate the correctness and completeness of the Web Service system. Any bugs found should be reported and fixed by the code owners.

943

944 **3.6.4.2 Roles**

945 User, Test Manager, Test System Administrator

946 **3.6.4.3 Artifacts**

947 The results should be recorded in Client Test Code, Test Scripts and Test Results.

948

949 Table 6 summaries the overview of each activities and the corresponding tasks, roles and
 950 artifacts under the activities.
 951

Activities	Tasks	Roles	Artifacts
<ul style="list-style-type: none"> ▪ Test functionality 	<ul style="list-style-type: none"> ▪ Test basic WS functionality ▪ Test for security ▪ Test UDDI functionality ▪ Test for SOAP intermediary capability 	<ul style="list-style-type: none"> ▪ Tester ▪ Test Designer 	<ul style="list-style-type: none"> ▪ Client test code ▪ Test scripts ▪ Test results
<ul style="list-style-type: none"> ▪ Integration test 	<ul style="list-style-type: none"> ▪ Test for conformance to WS-I ▪ Perform interoperability test based on various scenarios ▪ Perform integration test based on various scenarios 	<ul style="list-style-type: none"> ▪ Tester ▪ Test Designer ▪ Test System Administrator 	<ul style="list-style-type: none"> ▪ Client test code ▪ Test scripts ▪ Test results
<ul style="list-style-type: none"> ▪ System test 	<ul style="list-style-type: none"> ▪ Check system functionality and response time under different degrees of load increases ▪ Check functionality and response time under different combinations of valid and invalid requests 	<ul style="list-style-type: none"> ▪ Tester ▪ Test Designer ▪ Test System Administrator 	<ul style="list-style-type: none"> ▪ Client test code ▪ Test scripts ▪ Test results
<ul style="list-style-type: none"> ▪ User acceptance test 	<ul style="list-style-type: none"> ▪ Run UAT test cases 	<ul style="list-style-type: none"> ▪ User ▪ Test Manager ▪ Test System Administrator 	<ul style="list-style-type: none"> ▪ Client test code ▪ Test scripts ▪ Test results

952
 953 Notes: WS stands for Web Services
 954

955 *Table 6: Overview of activities, tasks, roles and artifacts in the Test Phase*

956
 957
 958 **3.7 Deployment Phase**

959 **3.7.1 Activity: Prepare deployment environment**

960 **3.7.1.1 Tasks**

- 961
- Set up and configure the hardware for Web Service deployment
- 962
- Set up and configure the software for Web Service deployment
- 963
- 964 The software may include application server, database, etc. The application server
- 965 should have a SOAP listener to support Web Services. Some Web Services may
- 966 need the SOAP handler to be configured.

967 **3.7.1.2 Roles**

968 System Engineer

969 **3.7.1.3 Artifacts**

970 Release Notes.

971

972 **3.7.2 Activity: Deploy Web Service**

973 **3.7.2.1 Tasks**

- 974
- Determine service URL
- 975 Web Service URL is unique and used to identify the Web Service and where it is
- 976 located.
- 977

- 978
- Prepare the deployment script
979 Deployment script is used to determine the steps of deployment. Although it is
980 different for different application server, most of them will include creation of directory,
981 copying files, shutting down and restarting the server.
- 982
- Deploy the Web Service
983 Execute the prepared deployment script.
- 984
- Generate WSDL file
985 After successfully deploying the Web Service, a WSDL file is needed to describe the
986 functions provided by the Web Service. WSDL can be created manually or by most
987 application servers, which will automatically generate the WSDL file after deployment.
988
989

990 **3.7.2.2 Roles**

991 Developer

992 **3.7.2.3 Artifacts**

993 WSDL File, Deployment Script.
994

995 **3.7.3 Activity: Test deployment**

996 **3.7.3.1 Tasks**

- 997
- Create (reuse) Web Service client code
998 The Web Service client code should be created by the developer during code and
999 debug phase.
- 1000
- Consume Web Service with the client code
1001 Because the functionality of the Web Service is properly tested, there is no need to
1002 test all the operations. To make sure the Web Service is properly deployed and
1003 configured, the best candidates of operations for invocation are the ones needed for
1004 database connection, configuration of SOAP handler or any other special features of
1005 the application server.
1006

1007 **3.7.3.2 Roles**

1008 Tester

1009 **3.7.3.3 Artifacts**

1010 Web Service Client Codes.
1011

1012 **3.7.4 Activity: Create end user support material**

1013 **3.7.4.1 Tasks**

- 1014
- Create end user support material
1015 The support material is needed to help the users to understand and use the Web
1016 Service. For example, an interoperability guide of the Web Service.

1017 **3.7.4.2 Roles**

1018 Developer

1019 **3.7.4.3 Artifacts**

1020 Interoperability Guide, User Guide, On-line Help, Tutorials and Training Material.

1021

1022 3.7.5 Activity: Publish Web Service

1023 3.7.5.1 Tasks

- 1024 • Identify the UDDI registry for publishing the Web Service
1025 Based on the requirements, decide whether a private or public UDDI registry is
1026 needed and the version of the UDDI Business Registry specifications to follow.
1027
- 1028 • Prepare the information needed for publishing
1029 The information may include key words for searching, description of Web Service,
1030 URL of WSDL file, etc.
1031
- 1032 • Publish the Web Service in the UDDI registry
1033 Normally, the UDDI registry will support the publishing via browser.
1034
- 1035 • Search the Web Service by key words after publishing
1036 Search the Web Service through browser provided by UDDI registry or tools provided
1037 by other vendors.

1038 3.7.5.2 Roles

1039 Developer

1040 3.7.5.3 Artifacts

1041 None.

1042

1043 Table 7 summaries the overview of each activities and the corresponding tasks, roles and
1044 artifacts under the activities.

1045

Activities	Tasks	Roles	Artifacts
<ul style="list-style-type: none"> ▪ Prepare deployment environment 	<ul style="list-style-type: none"> ▪ Set up and configure hardware ▪ Set up and configure software 	<ul style="list-style-type: none"> ▪ System Engineer 	<ul style="list-style-type: none"> ▪ Release Notes
<ul style="list-style-type: none"> ▪ Deploy WS 	<ul style="list-style-type: none"> ▪ Determine service URL ▪ Prepare deployment script ▪ Deploy WS ▪ Generate WSDL 	<ul style="list-style-type: none"> ▪ Developer 	<ul style="list-style-type: none"> ▪ WSDL file ▪ Deployment script
<ul style="list-style-type: none"> ▪ Test deployment 	<ul style="list-style-type: none"> ▪ Create (reuse) client code ▪ Consume WS with client code 	<ul style="list-style-type: none"> ▪ Tester 	<ul style="list-style-type: none"> ▪ Client codes
<ul style="list-style-type: none"> ▪ Create end user support material 	<ul style="list-style-type: none"> ▪ Create support material 	<ul style="list-style-type: none"> ▪ Developer 	<ul style="list-style-type: none"> ▪ Interoperability guide ▪ User guide ▪ On-line help ▪ Tutorials ▪ Training material
<ul style="list-style-type: none"> ▪ Publish WS 	<ul style="list-style-type: none"> ▪ Identify UDDI registry for publishing ▪ Prepare information for publishing ▪ Publish in UDDI registry ▪ Search by key words after publishing 	<ul style="list-style-type: none"> ▪ Developer 	<ul style="list-style-type: none"> ▪ --

1046

1047 Notes: WS stands for Web Services

1048

1049 *Table 7: Overview of activities, tasks, roles and artifacts in the Deployment Phase*

1050

1051

1052

4 References

1053

1. "Rational Unified Process", Version 2003.06.00.65, IBM-Rational Software.

1054

1055

2. "Rational Unified Process for Developing Web Services", Version 1.0, Java Smart Services Laboratory and Rational Software Pte. Ltd., Aug 2003.

1056

1057

Appendix A. Acknowledgments

1058

1059

The following individuals were members of the committee during the development of this documentation:

1060

- Ravi Shankar, CrimsonLogic Pte. Ltd.

1061

- Jagdip Talla, CrimsonLogic Pte. Ltd.

1062

- Andy Tan, Individual

1063

- Roberto Pascual, The Infocomm Development Authority of Singapore

1064

Appendix B. Revision History

Rev	Date	By Whom	What
wd-01	2004-09-30	Lai Peng CHAN Chai Hong ANG	Initial version
-	2004-12-23	Chai Hong ANG	Split the document into two
wd-01a	2005-05-24	Chai Hong ANG Puay Siew TAN Han Boon LEE	<ul style="list-style-type: none"> ▪ Remove Section 2.1 Terminology, Section 2.2 Concepts and Section 2.2.1 Web Service and combined them as Section 2.1 Objective ▪ Renumber Section 2.2.2 to Section 2.2 ▪ Renumber Section 2.2.3 to 2.3. The rest of the sub-sections are renumbered accordingly ▪ Added Table 1 as summary for phase and its assigned roles ▪ Section 2.2.4 Activity and Section 2.2.6 Artifact are moved into Section 2.5 Glossary ▪ Renumber Section 2.2.5 to 2.5 and put them into table format ▪ Section 3 is renamed as Web Service Implementation Methodology ▪ Removed Section 3.1 ▪ Renumber Section 3.1.1 to 3.1 ▪ Renumber Section 3.1.2 to 3.2. The rest of the sub-sections are renumbered accordingly ▪ Tables are added into each phase for summary ▪ Removed Normative and Non-Normative from Section 4
wd-01b	2005-06-02	Chai Hong ANG Prof. Marc Haines	<ul style="list-style-type: none"> ▪ Edited based on Prof. Haines' comments
wd-PublicReviewDraft1.0	2005-08-29	Eng Wah LEE	<ul style="list-style-type: none"> ▪ Added Prof. Marc Haines as Editor and renamed the document to Public Review Draft 1.0
wd-PublicReviewDraft1.0	2005-09-05	Eng Wah LEE	<ul style="list-style-type: none"> ▪ Removed the phrase "This document is updated periodically on no particular schedule. Send comments to the editor" under Status as this document is now in a stable version ready for public review.

1067

Appendix C. Notices

1068 OASIS takes no position regarding the validity or scope of any intellectual property or other
1069 rights that might be claimed to pertain to the implementation or use of the technology
1070 described in this document or the extent to which any license under such rights might or might
1071 not be available; neither does it represent that it has made any effort to identify any such
1072 rights. Information on OASIS's procedures with respect to rights in OASIS specifications can
1073 be found at the OASIS website. Copies of claims of rights made available for publication and
1074 any assurances of licenses to be made available, or the result of an attempt made to obtain a
1075 general license or permission for the use of such proprietary rights by implementors or users
1076 of this specification, can be obtained from the OASIS Executive Director.

1077 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1078 applications, or other proprietary rights which may cover technology that may be required to
1079 implement this specification. Please address the information to the OASIS Executive Director.

1080 **Copyright © OASIS Open 2005. All Rights Reserved.**

1081 This document and translations of it may be copied and furnished to others, and derivative
1082 works that comment on or otherwise explain it or assist in its implementation may be
1083 prepared, copied, published and distributed, in whole or in part, without restriction of any kind,
1084 provided that the above copyright notice and this paragraph are included on all such copies
1085 and derivative works. However, this document itself does not be modified in any way, such as
1086 by removing the copyright notice or references to OASIS, except as needed for the purpose
1087 of developing OASIS specifications, in which case the procedures for copyrights defined in
1088 the OASIS Intellectual Property Rights document must be followed, or as required to translate
1089 it into languages other than English.

1090 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1091 successors or assigns.

1092 This document and the information contained herein is provided on an "AS IS" basis and
1093 OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT
1094 LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL
1095 NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY
1096 OR FITNESS FOR A PARTICULAR PURPOSE.

1097