



Web Service Implementation Methodology

Public Review Draft, 6 July 2005

Document identifier:

fwsim-1.0-guidelines-doc-wd-01b.doc

Location:

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

Editors:

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

Contributors:

Marc HAINES, individual <mhaines@uwm.edu>
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:

This document is updated periodically on no particular schedule. Send comments to the editor.
Committee members should send comments on this specification to the fwsim-ims@lists.oasis-open.org list. Others should subscribe to and send comments to the fwsim-comment@lists.oasis-open.org list. To subscribe, send an email message to fwsim-comment-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/fwsim/>).

46 Table of Contents

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

84	3.2.5.1 Tasks	17
85	3.2.5.2 Roles	18
86	3.2.5.3 Artifacts.....	18
87	3.3 Analysis Phase.....	18
88	3.3.1 Activity: Select a technology platform as implementation framework.....	18
89	3.3.1.1 Tasks	18
90	3.3.1.2 Roles	19
91	3.3.1.3 Artifacts.....	19
92	3.3.2 Activity: Define a candidate architecture for the Web Service.....	19
93	3.3.2.1 Tasks	19
94	3.3.2.2 Roles	19
95	3.3.2.3 Artifacts.....	19
96	3.3.3 Activity: Decide on the granularity of the Web Service.....	19
97	3.3.3.1 Tasks	19
98	3.3.3.2 Roles	20
99	3.3.3.3 Artifacts.....	20
100	3.3.4 Activity: Identify reusable Web Services.....	20
101	3.3.4.1 Tasks	20
102	3.3.4.2 Roles	20
103	3.3.4.3 Artifacts.....	20
104	3.3.5 Activity: Identify service interface for new Web Services	20
105	3.3.5.1 Tasks	20
106	3.3.5.2 Roles	20
107	3.3.5.3 Artifacts.....	20
108	3.3.6 Activity: Prepare Test Cases for Performance Test	21
109	3.3.6.1 Task.....	21
110	3.3.6.2 Roles	21
111	3.3.6.3 Artifacts.....	21
112	3.3.7 Activity: Prepare Test Cases for Integration / Interoperability Test	21
113	3.3.7.1 Task.....	21
114	3.3.7.2 Roles	21
115	3.3.7.3 Artifacts.....	21
116	3.3.8 Activity: Prepare Test Cases for Functional Test	21
117	3.3.8.1 Task.....	21
118	3.3.8.2 Roles	21
119	3.3.8.3 Artifacts.....	21
120	3.3.9 Activity: Testbed preparation	21
121	3.3.9.1 Task.....	21
122	3.3.9.2 Roles	22
123	3.3.9.3 Artifacts.....	22
124	3.4 Design Phase	23
125	3.4.1 Activity: Transform signatures of reusable Web Services	23

126	3.4.1.1 Tasks	23
127	3.4.1.2 Roles	23
128	3.4.1.3 Artifacts.....	23
129	3.4.2 Activity: Refine service interface of the new Web Service.....	23
130	3.4.2.1 Tasks	23
131	3.4.2.2 Roles	23
132	3.4.2.3 Artifacts.....	23
133	3.4.3 Activity: Design Web Service.....	24
134	3.4.3.1 Tasks	24
135	3.4.3.2 Roles	24
136	3.4.3.3 Artifacts.....	24
137	3.4.4 Activity: Refine Test Cases for Functional Test.....	24
138	3.4.4.1 Task.....	24
139	3.4.4.2 Roles	24
140	3.4.4.3 Artifacts.....	24
141	3.5 Coding Phase.....	25
142	3.5.1 Activity: Construct Web Service code.....	25
143	3.5.1.1 Tasks	25
144	3.5.1.2 Roles	25
145	3.5.1.3 Artifacts.....	25
146	3.5.2 Activity: Construct Web Service client code	25
147	3.5.2.1 Tasks	25
148	3.5.2.2 Roles	26
149	3.5.2.3 Artifacts.....	26
150	3.5.3 Activity: Unit Test Web Service.....	26
151	3.5.3.1 Tasks	26
152	3.5.3.2 Roles	26
153	3.5.3.3 Artifacts.....	26
154	3.6 Test Phase	26
155	3.6.1 Activity: Test functionality of Web Service.....	27
156	3.6.1.1 Tasks	27
157	3.6.1.2 Roles	27
158	3.6.1.3 Artifacts.....	27
159	3.6.2 Activity: Integration Test on the Web Service.....	28
160	3.6.2.1 Tasks	28
161	3.6.2.2 Roles	28
162	3.6.2.3 Artifacts.....	28
163	3.6.3 Activity: System Test on the Web Service.....	28
164	3.6.3.1 Tasks	28
165	3.6.3.2 Roles	28
166	3.6.3.3 Artifacts.....	28
167	3.6.4 Activity: User Acceptance Test on the Web Service	28

168	3.6.4.1 Tasks	28
169	3.6.4.2 Roles	29
170	3.6.4.3 Artifacts.....	29
171	3.7 Deployment Phase.....	30
172	3.7.1 Activity: Prepare deployment environment.....	30
173	3.7.1.1 Tasks	30
174	3.7.1.2 Roles	30
175	3.7.1.3 Artifacts.....	30
176	3.7.2 Activity: Deploy Web Service.....	30
177	3.7.2.1 Tasks	30
178	3.7.2.2 Roles	31
179	3.7.2.3 Artifacts.....	31
180	3.7.3 Activity: Test deployment.....	31
181	3.7.3.1 Tasks	31
182	3.7.3.2 Roles	31
183	3.7.3.3 Artifacts.....	31
184	3.7.4 Activity: Create end user support material.....	31
185	3.7.4.1 Tasks	31
186	3.7.4.2 Roles	31
187	3.7.4.3 Artifacts.....	32
188	3.7.5 Activity: Publish Web Service	32
189	3.7.5.1 Tasks	32
190	3.7.5.2 Roles	32
191	3.7.5.3 Artifacts.....	32
192	4 References.....	33
193	Appendix A. Acknowledgments.....	34
194	Appendix B. Revision History	35
195	Appendix C. Notices	36
196		

List of Figures

198	Figure 1: Web Service Implementation Lifecycle	11
199	Figure 2: The “V” Model incorporates the Web Services specific Interoperability test.....	15
200	Figure 3: Relationship between phase, activities, tasks, roles and artifacts	16
201		

202

List of Tables

203	Table 1: Mapping between phases and roles assigned	12
204	Table 2: Overview of activities, tasks, roles and artifacts in the Requirements Phase	18
205	Table 3: Overview of activities, tasks, roles and artifacts in the Analysis Phase	23
206	Table 4: Overview of activities, tasks, roles and artifacts in the Design Phase	24
207	Table 5: Overview of activities, tasks, roles and artifacts in the Coding Phase	26
208	Table 6: Overview of activities, tasks, roles and artifacts in the Test Phase	30
209	Table 7: Overview of activities, tasks, roles and artifacts in the Deployment Phase	32
210		
211		

212

1 Introduction

213

1.1 Purpose

214

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

220

221

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

222

223

224

1.2 Target Audience

225

The target audiences are likely to be:

226

227

- Project Managers

228

This document provides a formal methodology for Web Services implementation, which can be used for management and control.

229

230

- Software Architects/Designers/Developers/Testers

231

This document identifies activities that are repeatable and which can be abide by, so as to ensure the quality of the software produced.

232

233

234

1.3 Scope

235

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

236

237

238

239

240

241

242

243

244

245

246

1.3.1 Not in Scope

247

This document does not define yet another novel software development methodology.

248

Instead, the Web Service implementation methodology highlights important features in the context of Web Services. The elements of the Web Service implementation methodology are based on existing agile software methodology and extend it by incorporating Web Service specific activities.

249

250

251

252

253

Also, it is not in the scope of this document to specifically address how each of these software development methodologies should be tailored to incorporate Web Service specific parts.

254

255

Examples are provided only to illustrate just one possible way of tailoring a specific agile development methodology for Web Service implementation.

256

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

257

258 This document does not intend to define a new software development methodology. Instead,
259 the Web Service Implementation Methodology leverages on an existing agile software
260 methodology and extend it by incorporating the Web Services specific activities.

261

262 This document also does not cover the detailed description or explanation of any of the
263 existing agile software development methodology nor does it recommend one particular agile
264 software development methodology over another.

265

2 Implementation Methodology Overview

266

2.1 Objective

267

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

271

272

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

273

274

275

276

277

2.2 Web Service Implementation Lifecycle

278

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

279

280

281

The Web Service implementation lifecycle typically includes the following phases:

282

1. Requirements Phase [see 2.3.1]

283

2. Analysis Phase [see 2.3.2]

284

3. Design Phase [see 2.3.3]

285

4. Coding Phase [see 2.3.4]

286

5. Test Phase [see 2.3.5]

287

6. Deployment Phase [see 2.3.6]

288

289

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

290

291

292

293

294

295

2.3 Phase

296

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

297

298

299

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

300

301

302

303

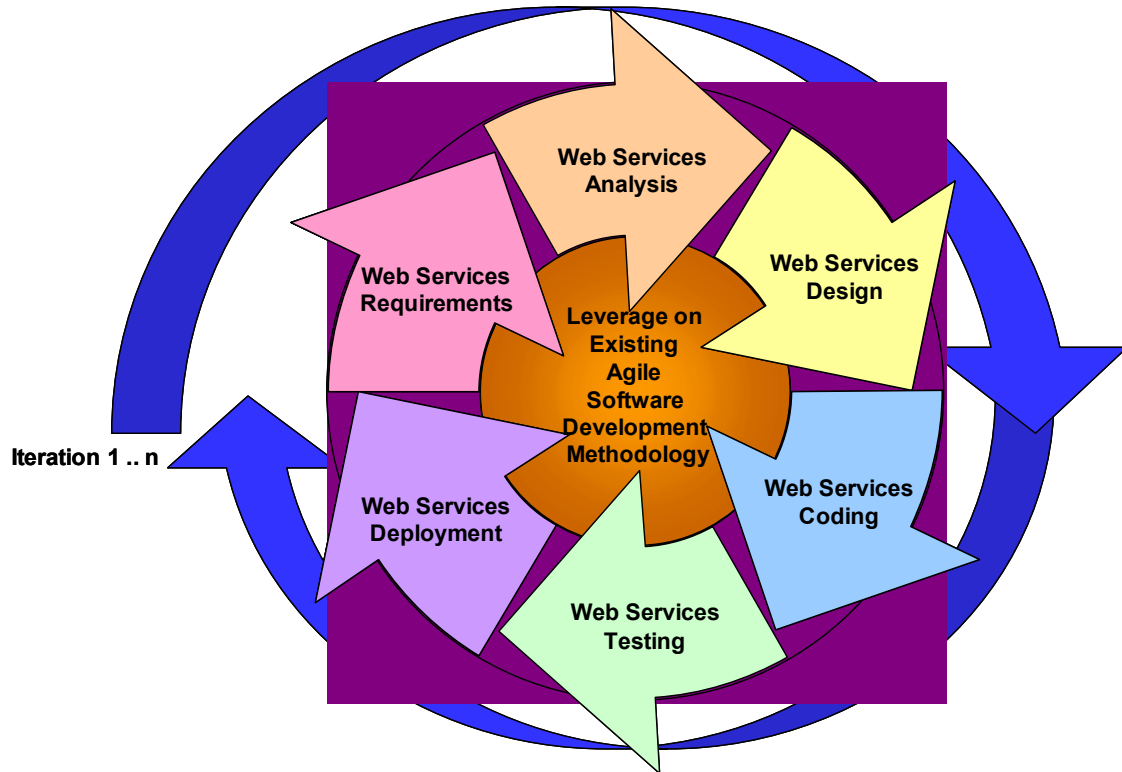


Figure 1: Web Service Implementation Lifecycle

305
306

307
308
309

310 2.3.1 Requirements Phase

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

314

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

319

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

324

325 2.3.2 Analysis Phase

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

332

333 2.3.3 Design Phase

334 The detailed design of Web Services is done in this phase. In this phase, the designers
335 should define the Web Service interface contract that has been identified in the analysis
336 phase. The defined Web Service interface contract should identify the elements and the
337 corresponding data types (possibly using a XML schema) as well as mode of interaction
338 between the Web Service and the client, for example, whether it should be
339 synchronous/asynchronous or RPC/Document style etc.
340

341 2.3.4 Coding Phase

342 The coding and debugging phase for Web Service implementation is essentially quite similar
343 to other software component-based coding and debugging phase. The main difference lies in
344 the creation of additional Web Service interface wrappers (to expose the components' public
345 APIs), generation of WSDLs and client stubs. Web Services in addition have to be deployed
346 to a Web Server/Application Server before the test clients can consume them.
347

348 The component developer and/or the tester should perform these activities.
349

350 2.3.5 Test Phase

351 For testing of Web Services, besides testing for functional correctness and completeness,
352 testers should also perform interoperability testing between different platforms and clients'
353 programs. Furthermore, performance testing has to be conducted to ensure that the Web
354 Services are able to withstand the maximum load and stress as specified in the non-functional
355 requirements specification. Other tasks like profiling of the Web Service application and
356 inspection of SOAP messages should also be done in this phase.
357

358 2.3.6 Deployment Phase

359 The purpose of the deployment phase is to ensure that the Web Service is properly deployed.
360 The phase will be executed after the service has been tested. The deployment of the Web
361 Service is platform specific. The service end points of the Web Service specifies where the
362 service is deployed and it needs to be identified and configured accordingly. The deployer
363 primary tasks are to ensure that the Web Service has been properly configured and managed
364 (e.g. version controlled, presetting of configuration files, packaged and loaded in the correct
365 location etc.) and running post-deployment tests to ensure that the Web Service is indeed
366 ready for use. Other optional tasks like specifying and registering the Web Service with an
367 UDDI registry may also be performed in this phase.
368

369 Table 1 summaries the overview of each phase against its' respective assigned roles.
370

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

371

372

Table 1: Mapping between phases and roles assigned

373

374

2.4 Role

375

Commonly defined roles in software development methodology include the following:

376

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

377

378

379

2.5 Glossary

380

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.

381

3 Web Service Implementation Methodology

382

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

388

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

393

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

402

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

410

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

416

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

423

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

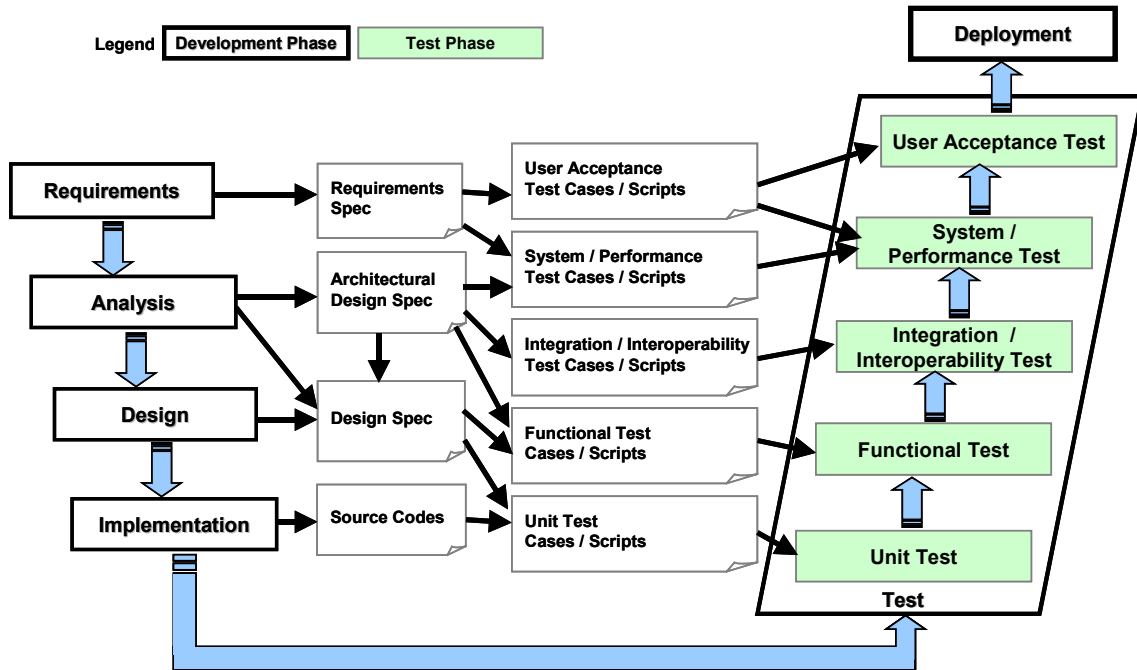
428

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

433

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.

434 In addition, for Web Service testing, a multitude of tests have to be conducted to ensure that
 435 the Web Service is developed according to its functional as well as non-functional
 436 requirements. Figure 2 illustrates using the “V” Model to perform these tests.
 437



438
 439

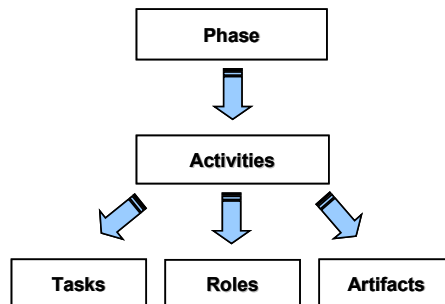
440 *Figure 2: The “V” Model incorporates the Web Services specific Interoperability test*

441
 442
 443
 444
 445
 446

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

447 3.1 Overview

448 The Web Service Implementation Lifecycle describes the phases a typical Web Service would
 449 undergo, from the identification of the need of the Web Service to the final deployment and
 450 usage by the end-users. The phases identified to be relevant in the Web Service
 451 Implementation Lifecycle are: requirements, analysis, design, code, test and deployment. In
 452 each of these phases, Web Service specific activities are carried out. These activities, as well
 453 as the roles and responsibilities, and the artifacts will be elaborated in the subsequent sub-
 454 sections. Figure 3 illustrates the above-mentioned relationship between phase, activities and
 455 their respective tasks, roles and artifacts.
 456



457

458

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

459 **3.2 Requirements Phase**

460 **3.2.1 Activity: Determine the need for Web Service**

461 **3.2.1.1 Tasks**

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

478 **3.2.1.2 Roles**

479 Architect, Requirements Analyst, Stakeholders, Project Manager

480 **3.2.1.3 Artifacts**

481 The results should be recorded in Business Requirement Specifications.

482

483 **3.2.2 Activity: Elicit Web Service requirements**

484 **3.2.2.1 Tasks**

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

497 **3.2.2.2 Roles**

498 Requirements Analyst, Architect, Test Manager

499 **3.2.2.3 Artifacts**

500 The results should be recorded in Requirement Specifications.

501

502 **3.2.3 Activity: Manage the Web Service requirements**

503 **3.2.3.1 Tasks**

- 504 • Based on the functional requirements categories, identify the Web Services and
505 establish the dependencies and priorities
506
- 507 • Create traceability matrices from the requirements to the identified Web Services
508 Traceability matrices help to track the requirements that have been taken care of by
509 the Web Services identified.
510
- 511 • Manage changes to the requirements

512 **3.2.3.2 Roles**

513 Requirements Analyst, Architect, Test Manager

514 **3.2.3.3 Artifacts**

515 The results should be recorded in Requirement Specifications.
516

517 **3.2.4 Activity: Model the usage scenarios**

518 **3.2.4.1 Tasks**

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

525 **3.2.4.2 Roles**

526 Requirements Analyst, Architect, Test Manager

527 **3.2.4.3 Artifacts**

528 The results should be recorded in Requirement Specifications.
529

530 **3.2.5 Activity: Prepare Test Cases for User Acceptance Test (UAT)** 531 **and System Test**

532 **3.2.5.1 Tasks**

- 533 • Write business scenario test case(s) based on the requirements gathered to be used
534 for UAT and System Test
535 Test case(s) can be derived from requirements. This is also a way to verify the
536 requirements when they are implemented.
537
- 538 • Build requirement validation matrix
539 The requirement validation matrix will include the requirements and a reference to the
540 test case(s) that will validate the requirement.
541

- 542 • Manage changes to the test cases when requirements changed

543 **3.2.5.2 Roles**

544 Requirements Analyst, Test Manager, Test Designer

545 **3.2.5.3 Artifacts**

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

547

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

550

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

551

552 Notes: WS stands for Web Services

553

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

555

556

557 **3.3 Analysis Phase**

558 **3.3.1 Activity: Select a technology platform as implementation framework**

559

560 **3.3.1.1 Tasks**

- 561 • Specify the Web Services standards that the implementation must adhere
- 562 Identify the Web Service standards based on the requirements and implementation
- 563 constraints. Consider issues like the standards compatibility, version of standards,
- 564 standards adoption in industry sector, and the organization approving the standards.
- 565
- 566 • Decide the technology platform for implementing Web Services

- 567 Choose a technology platform that is suitable for implementation. E.g. dotNet or Java
568 platform.
569
- 570 • Decide the technology platform for hosting Web Services
571 Based on implementation constrains and considerations for standards support and
572 interoperability requirements, choose the appropriate hosting platform for Web
573 Services.
574
 - 575 • Decide the IDE tools used to develop Web Services
576 Available options include commercial vendor's IDE tools, open source IDE tools.
577 Normally, the selection of IDE is tied together with the implementation platforms.

578 **3.3.1.2 Roles**

579 Architect

580 **3.3.1.3 Artifacts**

581 The results should be recorded in Software Architecture Specifications.
582

583 **3.3.2 Activity: Define a candidate architecture for the Web Service**

584 **3.3.2.1 Tasks**

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

595 **3.3.2.2 Roles**

596 Architect

597 **3.3.2.3 Artifacts**

598 The results should be recorded in Software Architecture Specifications.
599

600 **3.3.3 Activity: Decide on the granularity of the Web Service**

601 **3.3.3.1 Tasks**

- 602 • Decide on the coarseness of the Web Service operations to be exposed
603 Set up criteria on the coarseness of Web Services operations. Its definition depends
604 on the usage scenarios and requirements.
605
- 606 • Identify and group functionality into the Web Service
607 Based on the requirements and criteria mentioned above, identify the functions that
608 are needed to group into the Web Services.
609
- 610 • Decide on the mechanisms to compose or aggregate functionality

611 In case there is a need to compose individual Web Services, choose and decide the
612 mechanism to implement the compositions.

613 **3.3.3.2 Roles**

614 Architect

615 **3.3.3.3 Artifacts**

616 The results should be recorded in Software Architecture Specifications.
617

618 **3.3.4 Activity: Identify reusable Web Services**

619 **3.3.4.1 Tasks**

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

630 **3.3.4.2 Roles**

631 Architect

632 **3.3.4.3 Artifacts**

633 The results should be recorded in Software Architecture Specifications.
634

635 **3.3.5 Activity: Identify service interface for new Web Services**

636 **3.3.5.1 Tasks**

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

644 **3.3.5.2 Roles**

645 Architect, Designer

646 **3.3.5.3 Artifacts**

647 Web Service Signature Specifications, XML schema.
648

649 **3.3.6 Activity: P prepare Test Cases for Performance Test**

650 **3.3.6.1 Task**

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

657 **3.3.6.2 Roles**

658 Test System Administrator, Test Designer

659 **3.3.6.3 Artifacts**

660 The results should be recorded in Test Plan – Performance Test.
661

662 **3.3.7 Activity: P prepare Test Cases for Integration / Interoperability**
663 **Test**

664 **3.3.7.1 Task**

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

668 **3.3.7.2 Roles**

669 Test Designer, Tester

670 **3.3.7.3 Artifacts**

671 The results should be recorded in Test Plan – Integration / Interoperability Test.
672

673 **3.3.8 Activity: P prepare Test Cases for Functional Test**

674 **3.3.8.1 Task**

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

677 **3.3.8.2 Roles**

678 Test Designer, Tester

679 **3.3.8.3 Artifacts**

680 The results should be recorded in Test Plan - Functional Test.
681

682 **3.3.9 Activity: Testbed preparation**

683 **3.3.9.1 Task**

- 684 • Set up testing environment that include hardware and software

685
686
687

- This environment may be similar to the production/live environment in terms of hardware, OS, Web Server/Application Server, etc.

688 **3.3.9.2 Roles**

689 Test System Administrator, Test Designer

690 **3.3.9.3 Artifacts**

691 The results should be recorded in Test Plan - Testbed.

692

693 Table 3 summaries the overview of each activities and the corresponding tasks, roles and
694 artifacts under the activities.
695

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

696
697

Notes: WS stands for Web Services

698

699

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

700

701

702 **3.4 Design Phase**

703 **3.4.1 Activity: Transform signatures of reusable Web Services**

704 **3.4.1.1 Tasks**

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

716 **3.4.1.2 Roles**

717 Designer

718 **3.4.1.3 Artifacts**

719 The results should be recorded in Design Specifications.
720

721 **3.4.2 Activity: Refine service interface of the new Web Service**

722 **3.4.2.1 Tasks**

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

731 **3.4.2.2 Roles**

732 Designer

733 **3.4.2.3 Artifacts**

734 The results should be recorded in Design Specifications.
735

736 3.4.3 Activity: Design Web Service

737 3.4.3.1 Tasks

- 738 • Use some form of modeling techniques to describe the internal structure of the Web
739 Service
740 The design of the internal structure needs to consider the receiving and pre-
741 processing of request, delegating of the request, processing of the request and
742 sending of the response. Existing modeling techniques such as UML, design
743 patterns could be applied to the design.
744
- 745 • Consider non-functional requirements (e.g. usability, reliability, performance,
746 scalability etc.) and design constraints (e.g. interoperability etc.)

747 3.4.3.2 Roles

748 Designer

749 3.4.3.3 Artifacts

750 The results should be recorded in Design Specifications.
751

752 3.4.4 Activity: Refine Test Cases for Functional Test

753 3.4.4.1 Task

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

756 3.4.4.2 Roles

757 Test Designer, Tester

758 3.4.4.3 Artifacts

759 The results should be recorded in Test Plan – Functional Test.
760
761

762 Table 4 summaries the overview of each activities and the corresponding tasks, roles and
763 artifacts under the activities.
764

Activities	Tasks	Roles	Artifacts
▪ Transform signatures of reusable WS	▪ Identify data type mapping if required ▪ Identify design patterns for mapping the re-used WS interface to the desired one	▪ Designer	▪ Design Specifications
▪ Refine service interface of new WS	▪ Refine WS interface signature ▪ Refine XML schema for message exchange	▪ Designer	▪ Design Specifications
▪ Design WS	▪ Use modeling techniques to describe internal structure of WS ▪ Consider non-functional requirements and design constraints	▪ Designer	▪ Design Specifications
▪ Refine test cases for Functional Test	▪ Refine functional test cases	▪ Test Designer ▪ Tester	▪ Test Plan – Functional Test

765
766 Notes: WS stands for Web Services
767

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

769
770

771 **3.5 Coding Phase**

772 **3.5.1 Activity: Construct Web Service code**

773 **3.5.1.1 Tasks**

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

786 **3.5.1.2 Roles**

787 Developer

788 **3.5.1.3 Artifacts**

789 Web Service Implementation Codes.
790

791 **3.5.2 Activity: Construct Web Service client code**

792 **3.5.2.1 Tasks**

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

815 Use the WSDL to generate client stubs, which can be used in the client code to
 816 invoke the methods provided by the Web Service.

817 3.5.2.2 Roles

818 Developer

819 3.5.2.3 Artifacts

820 Web Service Client Codes.
 821

822 3.5.3 Activity: Unit Test Web Service

823 3.5.3.1 Tasks

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

826 3.5.3.2 Roles

827 Developer

828 3.5.3.3 Artifacts

829 Unit Test Scripts.
 830
 831

832 Table 5 summaries the overview of each activities and the corresponding tasks, roles and
 833 artifacts under the activities.
 834

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

835
 836 Notes: WS stands for Web Services
 837

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

839
 840

841 3.6 Test Phase

842 For Web Services, additional tests may be conducted to ensure that the Web Services are
 843 interoperable, secured and scalable.
 844

845 Interoperability is an issue in Web Services because the standards governing Web Services
 846 are still evolving. Furthermore, different vendors that implement these specifications may
 847 interpret and comply with these specifications differently. Currently there is an effort by Web
 848 Services Interoperability Organization (WS-I) to recommend basic profiles to minimise these
 849 incompatibilities. The aim of conducting interoperability tests is to ensure that these
 850 recommendations are followed and the Web Service developed will interoperate with other
 851 Web Services and products without problems.
 852

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

860
861 Web Services introduce special security issues e.g. in privacy, message integrity,
862 authentication and authorization. Tests have to be conducted to ensure that these security
863 requirements have been fulfilled. However, security schemes could complicate the process of
864 testing and debugging Web Service basic functionality. For example, non-intrusive monitors
865 are often used in functional testing but encrypted traffic presents an obvious complication to
866 this approach to testing.
867

868 **3.6.1 Activity: Test functionality of Web Service**

869 **3.6.1.1 Tasks**

870 • Testing basic Web Service functionality
871 The Web Service should respond correctly to requests from their clients. The format
872 of the SOAP message should be in compliance with the specifications. WSDL files,
873 which contain metadata about Web Services' interfaces, should be in compliance with
874 the WSDL specifications published by W3C. Perform fault checking to see how it
875 handles unexpected input. The test scripts and data prepared in the earlier phases
876 are executed in this activity. The test results should be recorded, and bugs found
877 should be reported to the code owners and fixed by them.

878

879 • Test for security
880 If a service requires a certain level of privacy, or if it requires that messages be
881 authenticated in a certain way, then specific tests are needed to ensure that these
882 security requirements are met. The test scripts and test data prepared in the earlier
883 phases should be executed in this activity. Any inadequacies that may lead to
884 possible security breaches should be reported and resolved by the code owner,
885 designer or architect.

886

887 • Test the UDDI functionality
888 If a service is registered to a registry server, perform registering of Web Service, then
889 write test clients to perform finding and binding of Web Service on the registry, and
890 then use the registry data to actually invoke the service. Test results from the test
891 scripts and data should be recorded and bugs should be fixed by the code owners.

892

893 • Test for SOAP intermediary capability
894 If particular SOAP message has one or more intermediaries along the message route
895 that take actions based upon the instructions provided to them in the header of the
896 SOAP message. Web Service SOAP intermediary testing must verify the proper
897 functionality of these intermediaries. Test results from the test scripts and data
898 should be recorded and bugs should be fixed by the code owners.

899 **3.6.1.2 Roles**

900 Tester, Test Designer

901 **3.6.1.3 Artifacts**

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

903

904 **3.6.2 Activity: Integration Test on the Web Service**

905 **3.6.2.1 Tasks**

- 906 • Test for conformance to Web Services Interoperability Organization (WS-I)
907 recommendations. Execute test scripts and data according to the test cases based
908 on the WS-I recommendations.
909
- 910 • Perform interoperability testing based on various scenarios
911 This is to highlight the interoperability issues of Web Services implementation. Refer
912 to Interoperability Guideline for the interoperability testing scenarios.
913
- 914 • Perform integration testing based on various scenarios
915 Based on the test cases prepared in the Analysis Phase, test scripts and test data,
916 which are prepared are executed and analyzed in this activity.

917 **3.6.2.2 Roles**

918 Tester, Test Designer, Test System Administrator

919 **3.6.2.3 Artifacts**

920 The results should be recorded in Client Test Code, Test Scripts and Test Results.
921

922 **3.6.3 Activity: System Test on the Web Service**

923 **3.6.3.1 Tasks**

- 924 • Check system functionality and response time under different degrees of load
925 increases
926 The test cases that are prepared in the earlier phases are executed in this activity.
927 The load increases can be sudden surges or gradual ramp-ups. The test results
928 should be analyzed to determine potential bottlenecks and if the system is scalable.
929
- 930 • Check functionality and response time under different combinations of valid and
931 invalid requests
932 The results from the test execution should be analyzed to determine if the system can
933 still render the expected quality of service as specified in the non-functional
934 requirement specifications.

935 **3.6.3.2 Roles**

936 Tester, Test Designer, Test System Administrator

937 **3.6.3.3 Artifacts**

938 The results should be recorded in Client Test Code, Test Scripts and Test Results.
939

940 **3.6.4 Activity: User Acceptance Test on the Web Service**

941 **3.6.4.1 Tasks**

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

946 **3.6.4.2 Roles**

947 User, Test Manager, Test System Administrator

948 **3.6.4.3 Artifacts**

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

950

951 Table 6 summaries the overview of each activities and the corresponding tasks, roles and
 952 artifacts under the activities.
 953

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

954
 955 Notes: WS stands for Web Services
 956

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

958
 959

960 3.7 Deployment Phase

961 3.7.1 Activity: Prepare deployment environment

962 3.7.1.1 Tasks

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

969 3.7.1.2 Roles

970 System Engineer

971 3.7.1.3 Artifacts

972 Release Notes.
 973

974 3.7.2 Activity: Deploy Web Service

975 3.7.2.1 Tasks

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

- 979
- 980
- 981
- 982
- 983
- 984
- 985
- 986
- 987
- 988
- 989
- 990
- 991
- Prepare the deployment script
Deployment script is used to determine the steps of deployment. Although it is different for different application server, most of them will include creation of directory, copying files, shutting down and restarting the server.
 - Deploy the Web Service
Execute the prepared deployment script.
 - Generate WSDL file
After successfully deploying the Web Service, a WSDL file is needed to describe the functions provided by the Web Service. WSDL can be created manually or by most application servers, which will automatically generate the WSDL file after deployment.

992 **3.7.2.2 Roles**

993 Developer

994 **3.7.2.3 Artifacts**

995 WSDL File, Deployment Script.

996

997 **3.7.3 Activity: Test deployment**

998 **3.7.3.1 Tasks**

- 999
- 1000
- 1001
- 1002
- 1003
- 1004
- 1005
- 1006
- 1007
- 1008
- Create (reuse) Web Service client code
The Web Service client code should be created by the developer during code and debug phase.
 - Consume Web Service with the client code
Because the functionality of the Web Service is properly tested, there is no need to test all the operations. To make sure the Web Service is properly deployed and configured, the best candidates of operations for invocation are the ones needed for database connection, configuration of SOAP handler or any other special features of the application server.

1009 **3.7.3.2 Roles**

1010 Tester

1011 **3.7.3.3 Artifacts**

1012 Web Service Client Codes.

1013

1014 **3.7.4 Activity: Create end user support material**

1015 **3.7.4.1 Tasks**

- 1016
- 1017
- 1018
- Create end user support material
The support material is needed to help the users to understand and use the Web Service. For example, an interoperability guide of the Web Service.

1019 **3.7.4.2 Roles**

1020 Developer

1021 **3.7.4.3 Artifacts**

1022 Interoperability Guide, User Guide, On-line Help, Tutorials and Training Material.
1023

1024 **3.7.5 Activity: Publish Web Service**

1025 **3.7.5.1 Tasks**

- 1026 • Identify the UDDI registry for publishing the Web Service
- 1027 Based on the requirements, decide whether a private or public UDDI registry is
- 1028 needed and the version of the UDDI Business Registry specifications to follow.
- 1029
- 1030 • Prepare the information needed for publishing
- 1031 The information may include key words for searching, description of Web Service,
- 1032 URL of WSDL file, etc.
- 1033
- 1034 • Publish the Web Service in the UDDI registry
- 1035 Normally, the UDDI registry will support the publishing via browser.
- 1036
- 1037 • Search the Web Service by key words after publishing
- 1038 Search the Web Service through browser provided by UDDI registry or tools provided
- 1039 by other vendors.

1040 **3.7.5.2 Roles**

1041 Developer

1042 **3.7.5.3 Artifacts**

1043 None.

1044
1045 Table 7 summaries the overview of each activities and the corresponding tasks, roles and
1046 artifacts under the activities.
1047

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"> ▪ --

1048
1049 Notes: WS stands for Web Services
1050

1051 *Table 7: Overview of activities, tasks, roles and artifacts in the Deployment Phase*

1052
1053

4 References

1054

1055

1056

1057

1058

1. "Rational Unified Process", Version 2003.06.00.65, IBM-Rational Software.
2. "Rational Unified Process for Developing Web Services", Version 1.0, Java Smart Services Laboratory and Rational Software Pte. Ltd., Aug 2003.

Appendix A. Acknowledgments

1059

1060 The following individuals were members of the committee during the development of this
1061 documentation:

1062 • Ravi Shankar, CrimsonLogic Pte. Ltd.

1063 • Jagdip Talla, CrimsonLogic Pte. Ltd.

1064 • Andy Tan, Individual

1065 • Roberto Pascual, The Infocomm Development Authority of Singapore

1066

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

Appendix C. Notices

1069

1070 OASIS takes no position regarding the validity or scope of any intellectual property or other
1071 rights that might be claimed to pertain to the implementation or use of the technology
1072 described in this document or the extent to which any license under such rights might or might
1073 not be available; neither does it represent that it has made any effort to identify any such
1074 rights. Information on OASIS's procedures with respect to rights in OASIS specifications can
1075 be found at the OASIS website. Copies of claims of rights made available for publication and
1076 any assurances of licenses to be made available, or the result of an attempt made to obtain a
1077 general license or permission for the use of such proprietary rights by implementors or users
1078 of this specification, can be obtained from the OASIS Executive Director.

1079 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1080 applications, or other proprietary rights which may cover technology that may be required to
1081 implement this specification. Please address the information to the OASIS Executive Director.

1082 **Copyright © OASIS Open 2005. All Rights Reserved.**

1083 This document and translations of it may be copied and furnished to others, and derivative
1084 works that comment on or otherwise explain it or assist in its implementation may be
1085 prepared, copied, published and distributed, in whole or in part, without restriction of any kind,
1086 provided that the above copyright notice and this paragraph are included on all such copies
1087 and derivative works. However, this document itself does not be modified in any way, such as
1088 by removing the copyright notice or references to OASIS, except as needed for the purpose
1089 of developing OASIS specifications, in which case the procedures for copyrights defined in
1090 the OASIS Intellectual Property Rights document must be followed, or as required to translate
1091 it into languages other than English.

1092 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1093 successors or assigns.

1094 This document and the information contained herein is provided on an "AS IS" basis and
1095 OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT
1096 LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL
1097 NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY
1098 OR FITNESS FOR A PARTICULAR PURPOSE.

1099