



FWSI IM Case Example using XP

Working Draft 03, 04 February 2005

Document identifier:

fwsim-0[1].0-xpcaseexample-doc-wd-03.doc

Location:

<http://www.oasis-open.org/apps/org/workgroup/fws/documents.php>

Editor:

Andy TAN, individual <andytan@intrinix.net>

Contributors:

Chai Hong ANG, Singapore Institute of Manufacturing Technology
<chang@SIMTech.a-star.edu.sg>

Lai Peng CHAN, Singapore Institute of Manufacturing Technology
<lpchan@SIMTech.a-star.edu.sg>

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

Puay Siew TAN, Singapore Institute of Manufacturing Technology
<pstan@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>

Zun Liang YIN, Singapore Institute of Manufacturing Technology
<zlyin@SIMTech.a-star.edu.sg>

Roberto PASCUAL, The Infocomm Development Authority of Singapore
<Roberto_B_Pascual@ida.gov.sg>

Abstract:

This document provides a guideline to prepare a case example to be included in the Implementation Methodology Guideline.

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 imsc@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 XXX TC web page (<http://www.oasis-open.org/committees/fwsu/>).

Table of Contents

43	1	Overview	4
44	2	Terminology Mapping.....	5
45	2.1	Implementation Lifecycle	5
46	2.2	Project Team Roles	5
47	2.2.1	Roles not defined in IM Guideline	6
48	2.3	Artifact Cross Reference.....	6
49	2.4	Concepts not defined in IM Guideline.....	6
50	2.4.1	Pair Programming	6
51	2.4.2	Continuous Integration	6
52	3	Implementation Phases.....	7
53	3.1	Planning: Requirement Phase & Analysis Phase	7
54	3.1.1	Determine the need for Web Services.....	7
55	3.1.2	Elicit Web Service requirements	7
56	3.1.3	Manage the Web Service requirement	7
57	3.1.4	Model the usage scenarios	7
58	3.1.5	Prepare test cases for User Acceptance Test and System test.....	8
59	3.1.6	Selecting a technology platform as implementation framework.....	8
60	3.1.7	Define a candidate structure architecture for the Web Services.....	8
61	3.1.8	Define on the granularity of the Web Services.....	8
62	3.1.9	Identify reusable Web Services.....	9
63	3.1.10	Identify service interface contract for new Web Services.....	9
64	3.1.11	Prepare Test Cases for Performance Test.....	9
65	3.1.12	Prepare Test Cases for Integration / Interoperability Test.....	9
66	3.1.13	Prepare Test Cases for Functional Test.....	10
67	3.1.14	Test bed preparation	10
68	3.1.15	Release Plan.....	10
69	3.2	Design Phase & Coding Phase	11
70	3.2.1	Transform signature of reusable Web Services.....	11
71	3.2.2	Refine service interface of the new Web Services.....	11
72	3.2.3	Design Web Services	11
73	3.2.4	Refine Test Cases for Functional Test	11
74	3.2.5	Code internal workings of Web Services.....	12
75	3.2.6	Write the Web Services Consumer Code.....	12
76	3.2.7	Unit Test Web Services	12
77	3.3	Test Phase	13
78	3.3.1	Test Functionality of Web Services	13
79	3.3.2	Integrated Test on the Web Services	13
80	3.3.3	System test on the Web Services	14
81	3.3.4	User Acceptance Test on the Web Services	14
82	3.4	Deployment Phase	14
83	3.4.1	Prepare deployment environment	14
84	3.4.2	Deploy Web Services	14

85	3.4.3	Test Deployment	15
86	3.4.4	Create end user support material.....	15
87	3.4.5	Publish Web Services.....	15
88	4	Reference	17
89		Appendix A. Acknowledgments	18
90		Appendix B. Revision History.....	19
91		Appendix C. Notices.....	20
92			

93

1 Overview

94 This case example illustrates how Extreme Programming (XP) can be adapted to incorporate the
95 Web Services specific activities described in the FWSI Implementation Methodology (IM)
96 Guideline.

97 We knew of XP in association with software development and wondered if it could solve the
98 problems in Web Services development. XP methodology could potentially deal with a number of
99 Web Services development problems. XP involves the customer (user), integrates teams,
100 generates code and most importantly divides the projects successfully between the developer,
101 the project manager and the customer.

102 Web Services project involves multi-disciplinary team made up of server-side programmers,
103 interface programmers, testers, project managers and users. Team members cannot work in
104 isolation because the decisions made by one affects the others. There are intersection and
105 overlapping of skills which make it impossible to set strict boundaries of responsibility.

106 XP is very suitable at getting programmers to communicate among themselves and with
107 customers.

108 Most software projects create deliverables for one platform at a time; however Web Services
109 project requires it to interoperate with disparate systems implemented in different platforms.

110 Web Services requires testing to account for the multiple systems interoperability. Often Web
111 Services need to be deployed rapidly and need to harmonies integration with new business
112 partners or customers.

113 Web Services development needs customers to set priorities, define the problem domain, and
114 make key decisions, because it is the customer who understands the process that the software is
115 trying to emulate. Web Services system and are often new to the organization. Thus, it is hard to
116 find an expert to consult. Customers look to their developers for guidance.

117

118

119 2 Terminology Mapping

120 This section explains the terminology used in XP and how it is related to terminology used in IM
121 Guideline. The aim is to provide a quick reference to help reader to cross reference terminology
122 used in XP to that used in IM Guideline.

123 2.1 Implementation Lifecycle

124 XP method defines the following phases: Planning, Design, Coding and Testing. These phases
125 are applied in Web Services development and implementation.

126 Following map XP phases to Web Services phases:

XP phases	IM Guideline Phases
Planning	Requirement Phase Analysis Phase
Design	Design Phase
Coding	Coding Phase
Testing	Testing Phase
Releases	Deployment Phase

127 Web Services, unlike traditional software is not meant for end-user consumption. Web Services
128 are pieces of business logic which have programmatic interfaces and it is through these
129 interfaces that developers can create new application systems.

130 XP method is extremely useful in Web Services implementation. XP method focuses on short
131 "iterations" of two or three weeks, and you select the work to be done in each of those iterations.
132 Your mission is to select work that will maximize the business value completed by the end of the
133 iteration.

134 2.2 Project Team Roles

135 Not know what you suppose to do will lead to disaster. Defining the role of each member is the
136 first steps. With roles and responsibilities for team members defined, projects run much more
137 smoothly and team morale improves.

138 Web Services development is different from typical XP projects and therefore the roles need to be
139 modified for Web Services.

140 Here are some of the main roles that have been suggested in the XP literature.

- 141 • The *manager* schedules the iteration meetings, ensures that the process is being
142 followed, provides reporting, and removes project obstacles.
- 143 • The *customer* writes and prioritizes user stories (think of stories as ideas for features)
144 and writes acceptance criteria. She has the authority to make decisions.
- 145 • The *coach* oversees the entire project to ensure that the team is following XP best
146 practices.
- 147 • The *tracker* tracks the teams' progress, helping them solve problems and warning the
148 manager of any potential problems.

- 149 • The *programmer* estimates stories, breaking them up into tasks and estimating tasks;
150 volunteers for stories; and writes unit tests.
- 151 • The *tester* helps the customer write acceptance criteria, writes and runs functional tests,
152 and reports test results.
- 153 • The *doomsayer*, or naysayer, is anyone on the team who senses a problem and brings it
154 to the attention of the team.

155 Following maps the XP roles to IM Guideline Roles

XP suggested Roles	IM Guideline Roles
Programmer	Requirement Analyst, Architect, Designer, Developer
Customer	Stakeholder
Manager, Coach, Tracker	Project Manager, Deployer
Tester	Tester, Test Designer, Test Manager

156 Each person can play more than one role and may change roles in different phases of the project.
157 It does not mean that you need 10 people to start a Web Services project.

158 **2.2.1 Roles not defined in IM Guideline**

159 The doomsayer role is symbolic and played by different people at different times.

160 **2.3 Artifact Cross Reference**

161 In XP, customer requirements are describe in the form of stories. Each story describes one thing
162 that the system needs to do. Each story must be understood well enough that the programmers
163 can estimate its difficulty.

164 Stories are similar to Requirements Specification or Use Cases but written as a narration for non-
165 technical audience. XP places the responsibility for writing stories squarely on the customer.
166 However, in Web Services projects customers are not the domain experts we need, so stories are
167 written primarily by a project manager. Customers still set priorities and aid in strategy
168 development, but this is not the traditional XP story writing practice.

169 On the other hand, XP doctrine does not define the artifact for the process. We will use the
170 terminology proposed in the IM Guideline.

171 **2.4 Concepts not defined in IM Guideline**

172 This section listed the concepts not defined in IM Guideline but is necessary in XP.

173 **2.4.1 Pair Programming**

174 Pair programming, one of the pillars of XP is important in Web Services development. We started
175 experimenting with Web Services development. First we tried pairing up interface programmer
176 with backend developer. We manage to solve interface problem faster and the quality of code is
177 improved. The programmers are happier and the morale of the team improved.

178 **2.4.2 Continuous Integration**

179 XP continuous integration fosters teamwork on a Web Services development project. XP
180 methodology is iterative and incremental.

181

182 **3 Implementation Phases**

183 This section describes the implementation phases using XP terminology and doctrine.

184 **3.1 Planning: Requirement Phase & Analysis Phase**

185 **3.1.1 Determine the need for Web Services**

186 An XP team plans and builds software in terms of "stories". For a project spanning a few months,
187 there may be 50 or 100 stories. Larger projects of course have more stories.

188 **3.1.1.1 Roles**

189 Customer, Manager

190 **3.1.1.2 Artifacts**

191 Stories, Business Requirements and Business Plan

192 **3.1.2 Elicit Web Service requirements**

193 Stories are individual stories about how the system needs to work. Stories collected in 3.1.1 are
194 described at a higher level and with the help of customer; the manager will break it down into sub
195 stories with focus on requirements. Manager translates these stories into technical specification.

196 **3.1.2.1 Roles**

197 Customer, Manager

198 **3.1.2.2 Artifacts**

199 Stories, Requirement Specification

200 **3.1.3 Manage the Web Service requirement**

201 The customer express what must be done in terms of stories. Project Manager documents these
202 stories. Manager translates these stories into technical specification.

203 **3.1.3.1 Roles**

204 Customer, Manager

205 **3.1.3.2 Artifacts**

206 Stories, Requirement Specification

207 **3.1.4 Model the usage scenarios**

208 Each story describes one thing that the system needs to do. Each story must be understood well
209 enough that the programmers can estimate its difficulty. The manager translates these stories
210 into technical specification.

211 **3.1.4.1 Roles**

212 Customer, Manager, Programmer

213 **3.1.4.2 Artifacts**
214 Stories, Requirement Specification

215 **3.1.5 Prepare test cases for User Acceptance Test and System test**
216 And each story must be testable.

217 **3.1.5.1 Roles**
218 Customer, Manager, Tester, Programmer

219 **3.1.5.2 Artifacts**
220 Test Plan – UAT and System Test

221 **3.1.6 Selecting a technology platform as implementation framework**
222 While many companies work on technologies on their own, we like to involve the customer. We
223 start by walking him through the various languages and frameworks that can be employed,
224 describing the benefits and drawbacks of each of them.

225 For example, it is worth the effort to provide a presentation on the advantages of an object-
226 oriented language, the need for clustering, and the relative merits of Oracle and SQL Server to
227 the customers. These are decisions that the customer should participate in making. Never
228 assume otherwise because the choice of technologies used in Web Services development will
229 affect the lifecycle of the Web Services, ongoing maintenance costs, and other real business
230 concerns.

231 **3.1.6.1 Roles**
232 Customer, Manager, Programmer

233 **3.1.6.2 Artifacts**
234 None

235 **3.1.7 Define a candidate structure architecture for the Web Services**
236 Once we are clear about what the customer is trying to achieve, we can discuss what direction
237 the customer should take and types of content to be offered as Web Services.

238 Several options will arise that the customer has expressed interest in. The next step is to evolve
239 these options into possible Web Services.

240 We can start to define a high-level architecture. Identify the components that expose functionality
241 as Web Services.

242 **3.1.7.1 Roles**
243 Customer, Manager, Tester, Programmer

244 **3.1.7.2 Artifacts**
245 Software Architecture Specification

246 **3.1.8 Define on the granularity of the Web Services**
247 We will start to decide on the coarseness of the Web Services operations to be exposed based
248 on the stories gathered from the customers.

249 Identify the group of services and group functionality into Web Services. Decide on the
250 mechanisms to compose or aggregate functionality.

251 **3.1.8.1 Roles**

252 Manager, Programmer

253 **3.1.8.2 Artifacts**

254 Software Architecture Specification

255 **3.1.9 Identify reusable Web Services**

256 Identify any of the existing Web Services can provide equivalent Web Services. If the functionality
257 can be offered by existing Web Services, then place a remark on the identified Web Services.

258 **3.1.9.1 Roles**

259 Manager, Programmer

260 **3.1.9.2 Artifacts**

261 Software Architecture Specification

262 **3.1.10 Identify service interface contract for new Web Services**

263 Those identified Web Services that do not fixed into any of the existing Web Services then new
264 Web Services are needed. Based on the requirements and usage model, identify its operation
265 and signature.

266 **3.1.10.1 Roles**

267 Programmer

268 **3.1.10.2 Artifacts**

269 Software Architecture Specification

270 **3.1.11 Prepare Test Cases for Performance Test**

271 Write procedure to test the performance.

272 **3.1.11.1 Roles**

273 Programmer, Tester, Manager

274 **3.1.11.2 Artifacts**

275 Test Plan – Performance Test

276 **3.1.12 Prepare Test Cases for Integration / Interoperability Test**

277 Write the procedures to tests integration and interoperability.

278 **3.1.12.1 Roles**

279 Programmer, Tester, Manager

280 **3.1.12.2 Artifacts**

281 Test Plan – Integration and Interoperability Test

282 **3.1.13 Prepare Test Cases for Functional Test**

283 Write functional test procedures. These test procedure are prepared to verify each stories
284 gathered in the earlier steps.

285 **3.1.13.1 Roles**

286 Programmer, Tester and Manager

287 **3.1.13.2 Artifacts**

288 Test Plan – Functional Test

289 **3.1.14 Test bed preparation**

290 At this stage, we need to setup a testing environment that includes hardware and software. This
291 environment is similar to the production/live environment in terms of hardware, OS, Web Server
292 and Application Server, etc.

293 Prepare a test system which can execute test script and capture data for analysis. This is to
294 enable automatic testing.

295 **3.1.14.1 Roles**

296 Tester

297 **3.1.14.2 Artifacts**

298 Test Plan – Test Bed

299 **3.1.15 Release Plan**

300 Generating a Release Plan is as much an exercise in customer relations as it is the writing of a
301 document. Web Services project requirements seem fairly loose to begin with, so simply
302 spending a day with the customer usually is not enough to get all the information you need. If you
303 want a happy customer, you need to firm up information in at least four key areas:

- 304 • What the customer is hoping to achieve through the Web Services
- 305 • What the best strategies are to achieve those goals
- 306 • What technical constraints apply to the target audience for the site
- 307 • What Web technologies are most appropriate

308 There are no easy answers to writing the release plan, but here is the process we have found
309 most effective in producing successful projects. Keep it short and cover only the important issues.
310 XP is not about generating documents.

311 **3.1.15.1 Roles**

312 Customer, Manager, Programmer, Tester, Tracker

313 **3.1.15.2 Artifacts**

314 Project Schedule

315 **3.2 Design Phase & Coding Phase**

316 **3.2.1 Transform signature of reusable Web Services**

317 We have found that Web Services design requires some closely followed best practices to further
318 this goal:

319 Prototypes are useful to test and verify some concept, algorithms or methods. Prototypes are
320 disposable and they are used for learning something and testing. Never use prototype code in
321 production code.

322 To identify problems early in the project, try to complete a few simple tasks before you attempt
323 anything complicated. Any problem will be much clearer and easier to resolve at this point.

324 Identify components that were developed and determine if it can be reused. Examine the data
325 structures and mapped to existing components if necessary.

326 **3.2.1.1 Roles**

327 Programmer

328 **3.2.1.2 Artifacts**

329 Software Design Specification

330 **3.2.2 Refine service interface of the new Web Services**

331 Programme code need to be reviewed regularly and enhance them if needed. Regular
332 housecleaning is also need.

333 **3.2.2.1 Roles**

334 Programmer

335 **3.2.2.2 Artifacts**

336 Software Design Specification

337 **3.2.3 Design Web Services**

338 Study the customer stories gathered in the Requirements and Analysis Phase. Write each story
339 on a stick pad. The Programmer or Designer make sure he understands what the customers
340 needs then draw up boxes of logical functions of Web Services. Then sticks each stories into the
341 most appropriate function boxes.

342 **3.2.3.1 Roles**

343 Programmer

344 **3.2.3.2 Artifacts**

345 Software Design Specification

346 **3.2.4 Refine Test Cases for Functional Test**

347 Review the test procedure prepared at section 2.1.13 is refined into detail test steps which
348 include test data to be tested.

349 **3.2.4.1 Roles**

350 Programmer, Tester, Manager

351 **3.2.4.2 Artifacts**

352 Test Plan – Functional Test

353 **3.2.5 Code internal workings of Web Services**

354 Coding is so much more than just sitting in front of the computer and typing. Coding is problem
355 solving and requires a great deal of attention and creativity. There are so many obstacles to
356 turning the design into code.

357 In Extreme Programming, the emphasis is on programming. We will focus on developing optimal
358 code and modular.

359 Pair programming is useful at this stage; the programmer working on the Web Services
360 Consumer Code will work with the programmer working on the Web Services Server code.

361 **3.2.5.1 Roles**

362 Programmer

363 **3.2.5.2 Artifacts**

364 Software source code

365 **3.2.6 Write the Web Services Consumer Code**

366 This task will be relatively easy as the programmer is very familiar with the Web Services
367 Interfaces. The programmer needs to focus on Web Services Client programming model to use.

368 **3.2.6.1 Roles**

369 Programmer

370 **3.2.6.2 Artifacts**

371 Software source code

372 **3.2.7 Unit Test Web Services**

373 At this stage, deploy the Web Services in local test environment and perform functional unit
374 testing. The emphasis is on the correctness of the functionality and the exceptions handling.

375 We can adopt the technique of “Test first by intention”. Prepare a set of test data to input to the
376 system and check against the output to see that it is correct. We need to perform data boundaries
377 check to see if the Web Services fail. Proceed to solve the problem when each test step failed.
378 When the test works, move on to the next.

379 We are trying to test everything that could possibly break in this object. We want to make sure
380 that every sections of the code are tested. For example, every if and case loop must be executed.
381 The Tester and Programmer can pair up to do the test together. Programmer focuses on
382 debugging and writing the code while the Tester writes the test scripts.

383 Write these test as script so that the tests can be repeated every time we need to change a
384 section of the code.

385 Automated testing is done for unit tests. These are small tests that allow programmer to test the
386 method in the objects on a pass or fail basis. Every public methods or function calls are tested.
387 These set of tests are useful when changes is made on the object. We can see immediately when
388 something breaks.

389 As Web Services systems can get larger and single changes in one place can caused havoc to
390 the whole system. It is very difficult debug in a system level so unit tests are a reliable way to test
391 and ensure that all the code are tested before deployment.

392 **3.2.7.1 Roles**

393 Tester

394 **3.2.7.2 Artifacts**

395 Test scripts

396 **3.3 Test Phase**

397 **3.3.1 Test Functionality of Web Services**

398 Functional tests focus on basic Web Services Functionality. As each component objects are
399 tested in the unit test, functional tests focus on the expected operations of the Web Services
400 using the respective components. It also test the interface of various components used to
401 implement Web Services functions.

402 Some projects have a legacy system they are replacing, and they can get their test data from the
403 legacy system. In this case, your job will be to select the legacy inputs and outputs you want
404 tested. Some projects use spreadsheets from the customer that provide the inputs and expected
405 outputs. Smart XP programmers will extract the information from the spreadsheet automatically
406 and read it into the system. Some projects use manually calculated values that are typed in by
407 someone on the team.

408 The format of SOAP messages are verified and should be in compliance with the specification.
409 Test results shall be automatically logged and compare with the expected.

410 The test procedures prepared in the earlier phase are carried out as check list to certify that the
411 system is functioning as specified.

412 **3.3.1.1 Roles**

413 Tester

414 **3.3.1.2 Artifacts**

415 Test Scripts

416 **3.3.2 Integrated Test on the Web Services**

417 Some customers give input numbers to the programmers and check the output by just reading it.
418 There is an important issue with this method. Checking the output of a computer is very error-
419 prone. It's easy to look at the numbers and decide that they look correct. It's also easy to be
420 wrong when you do that. It is better to have the expected answers up front, even if they have to
421 be computed by hand.

422 At this stage, business partners are invited to participate in the tests. Test for conformance to
423 Web Services Interoperability Organization (WS-I) is recommended. Test scripts are prepared
424 and data according to the test cases based on the WS-I recommendations.

425 Tests are performed based on various scenarios. The tests first focus on passed or fail based
426 and progresses to test for performance.

427 **3.3.2.1 Roles**

428 Tester

429 **3.3.2.2 Artifacts**

430 Test Scripts

431 **3.3.3 System test on the Web Services**

432 All tests in an XP project must be automated. It gives you the ability to move very rapidly, and to
433 change your requirements any time you need to. This means that the code will be changing
434 rapidly. The only way to move rapidly with confidence is to have a strong network of tests that
435 ensure that changes to the system do not break things that already work.

436 System test checks for overall system functionality which include all the components that made up
437 the system. In this stage, the system response time is checked under different expected load.

438 Data are captured to analyse to determine potential bottlenecks and if the system is scalable to
439 meet these unexpected demand.

440 **3.3.3.1 Roles**

441 Tester

442 **3.3.3.2 Artifacts**

443 Test Scripts

444 **3.3.4 User Acceptance Test on the Web Services**

445 Acceptance tests really need to be available in the same iteration as the story is scheduled. You
446 want to score the development based on getting stories done, and the only way to know if they
447 are really done is to run the tests.

448 **3.3.4.1 Roles**

449 Customer, Manager, Programmer, Tester, Tracker

450 **3.3.4.2 Artifacts**

451 Test Scripts, Test Procedures and Check list

452 **3.4 Deployment Phase**

453 **3.4.1 Prepare deployment environment**

454 One of the most important things you can do on an XP project is to release early and release
455 often. Each independent module is released to allow customer to use. However care must be
456 taken to make sure that each release is properly tested.

457 Each release provides an opportunity for the customer to try out the application and give you
458 feedback. We would learn a lot about what customer really wanted.

459 **3.4.1.1 Roles**

460 Manager, Programmer, Tracker, Customer

461 **3.4.1.2 Artifacts**

462 Deployment Plan

463 **3.4.2 Deploy Web Services**

464 At this stage, we are ready for deployment. The services URL is gathered from the customer and
465 verifies that it is unique.

466 Prepare a deployment script. Deployment script is used to determine the steps of deployment.
467 Deployment scripts are also used for future redeployment of systems due to hardware or
468 operating system upgrade as well as disaster recovery. XP is light on documentation and
469 deployment script defined the installation procedure in detail.

470 **3.4.2.1 Roles**

471 **3.4.2.2 Artifacts**

472 **3.4.3 Test Deployment**

473 The Web Service is ready to put into operation. User accounts are setup and production data are
474 imported into the system.

475 The functionality of the Web Services is properly tested and there is no need to test all the
476 operation. This stage will focus on actual client consumption of services. The invocation of clients
477 services from participating business partners or users are tested and the return results are
478 checked against results obtained from current method.

479 If there are discrepancies, the data are checked to determine the source of error. It could be due
480 to error data generated due to current method or there are errors in the Web Services.

481 **3.4.3.1 Roles**

482 Programmer

483 **3.4.3.2 Artifacts**

484 WSDL, Deployment Script, Software Object Code

485 **3.4.4 Create end user support material**

486 User manual are generated to help users to understand and use the Web Services. For example,
487 the user manual describes how to invoke the various services.

488 Managing the assets of the project is also important. Assets includes source codes, all test
489 scripts, diagrams, test procedures, customer stories must be properly documented and archived

490 **3.4.4.1 Roles**

491 Customer, Manager, Tester, Programmer

492 **3.4.4.2 Artifacts**

493 User Guide, Installation Manual, Training Material, On-line help

494 **3.4.5 Publish Web Services**

495 At this stage, depending on the requirement, the Web Services is ready to be published on the
496 UDDI registry. The customers will decide whether a private or public UDDI registry is needed and
497 the version of the UDDI Business Registry specification to follow.

498 Prepare the information needed for publishing. Information may include key words for searching,
499 description of Web Services URL of WSDL file, etc.

500 Publish the Web Services to the UDDI registry. Normally this is done through the Web Browser.
501 Verify that it is properly registered by performing a search through browser or tools provided by
502 the vendors.

503 **3.4.5.1 Roles**

504 Programmer

505 **3.4.5.2 Artifacts**

506 None

507

508 **4 Reference**

509

510

511

512 **Appendix A. Acknowledgments**

513 The following individuals were members of the committee during the development of this
514 specification.

515 Andy TAN, Individual <andytan@intrinix.net>
516 Chai Hong ANG, Singapore Institute of Manufacturing Technology
517 <chang@SIMTech.a-star.edu.sg>
518 Eng Wah LEE, Singapore Institute of Manufacturing Technology
519 <ewlee@SIMTech.a-star.edu.sg>
520 Puay Siew TAN, Singapore Institute of Manufacturing Technology
521 <pstan@SIMTech.a-star.edu.sg>
522 Yushi CHENG, Singapore Institute of Manufacturing Technology
523 <ycheng@SIMTech.a-star.edu.sg>
524 Xingjian XU, Singapore Institute of Manufacturing Technology
525 <xjxu@SIMTech.a-star.edu.sg>
526 Zun Liang YIN, Singapore Institute of Manufacturing Technology
527 <zlyin@SIMTech.a-star.edu.sg>
528 Roberto PASCUAL, The Infocomm Development Authority of Singapore
529 <Roberto_B_Pascual@ida.gov.sg>
530 JAGDIP Talla, CrimsonLogic Pte Ltd <jagdip@crimsonlogic.com>
531 RAVI SHANKAR Narayanan Nair, CrimsonLogic Pte Ltd <ravishankar@crimsonlogic.com>
532 Marc HAINES, individual <mhaines@uwm.edu>

533 **Appendix B. Revision History**

Rev	Date	By Whom	What
Draft 01	25 Nov 2004	Andy Tan	Initial Draft
Draft 02	16 Dec 2004	Andy Tan	Corrected spelling errors and completed roles and artifacts.
Draft 03	04 Feb 2005	Andy Tan	Using the Case Example Guide template

534

535 **Appendix C. Notices**

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

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

548 Copyright © OASIS Open 2002. *All Rights Reserved.*

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

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

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