Autor: María Luz Blanco

Director: Javier Blanqué

Universidad Nacional de Luján
Int. Ruta 5 y 7
6700 Luján, Buenos Aires
República Argentina
Año 2007

METODOLOGÍAS DE DESARROLLO DE SOFTWARE APLICADAS A SOA (*)

María Luz Blanco
marialuzblanco@gmail.com

RESUMEN

El objetivo de este trabajo es conocer acerca de la utilización de las metodologías de desarrollo de software en proyectos con arquitecturas orientadas a servicios.

Se investiga, en primera instancia, las metodologías de desarrollo existentes independientemente del tipo de arquitectura subyacente. Éstas van desde las tradicionales, las ágiles hasta las híbridas.

Luego se estudia brevemente en qué consiste una arquitectura orientada a servicios. Esto se realiza para poder comprender mejor el aporte y la necesidad de determinadas metodologías en proyectos de este tipo. Se dará un breve detalle acerca de aquellas que surgieron específicamente para proyectos con SOA.

Por último se obtiene la opinión de personas que participan en proyectos de desarrollo de software, acerca de su experiencia con metodologías y SOA.

(*) SOA: Service Oriented Architectures o Arquitecturas Orientadas a Servicios

Palabras clave:

SOA. Metodologías. Servicios. Desarrollo de software. Metodologías Ágiles. Metodologías Tradicionales.

TABLA DE CONTENIDOS

AGRADECIMIENTOS 1

ORGANIZACIÓN DEL TRABAJO 2

1. INTRODUCCIÓN 3
2. ANTECEDENTES 4
3. METODOLOGÍAS DE DESARROLLO DE SOFTWARE 5

3.1 METODOLOGÍAS TRADICIONALES DE DESARROLLO DE SOFTWARE 5

3.1.1 PROCESO CASCADA (Waterfall Process). 5
3.1.1.1 Características 5
3.1.1.2 Ventajas 6
3.1.1.3 Limitaciones 6
3.1.2 PROCESO ITERATIVO 7
3.1.2.1 Características 7
3.1.2.2 Ventajas 8
3.1.2.3 Limitaciones 9
3.1.3 PROCESO ESPIRAL 10
3.1.3.1 Características 10
3.1.3.2 Pasos involucrados en el desarrollo en Espiral. 11
3.1.3.3 Ventajas 12
3.1.3.4 Limitaciones 12
3.1.4 UP - RUP 13
3.1.4.1 Características 14
3.1.4.2 Fases 15
3.1.4.3 Flujos de trabajo 19
3.1.4.4 Buenas Prácticas. 21
3.1.4.5 Ventajas. 23
3.1.4.6 Limitaciones. 23
3.1.5 Jackson Development Methods 29
3.1.5.1 Jackson Structured Programming (JSP) 30
3.1.5.2 Jackson System Development (JSD) 32

3.2 METODOLOGÍAS ÁGILES DE DESARROLLO DE SOFTWARE 35

3.2.1 Introducción. 35
3.2.2 Surgimiento 35
3.2.3 El Manifiesto Ágil. 36
3.2.4 Características. 37
3.2.5 Comparación con las metodologías “tradicionales”. 39
3.2.6 Ventajas. 40
3.2.7 Limitaciones. 42
3.2.8 Metodologías Ágiles: 44
3.2.8.1 Extreme Programming 44
3.2.8.2 Métodos Crystal 53
3.2.8.3 Scrum 57
3.2.8.4 Dynamic Systems Development Method 63
3.2.8.5 Lean Programming 67
3.2.8.6 Feature-Driven Development 70
3.2.8.7 Iconix 74
3.2.9 Comparaciones: 77

4. ARQUITECTURAS ORIENTADAS A SERVICIOS 79

4.1 Introducción 79

4.2 Antecedentes *79*
4.3 Concepto *81*
4.4 Componentes *82*
4.5 Roles. *94*
4.6 SOA y otras arquitecturas. *94*
4.7 Ventajas. *95*
4.8 Desventajas *99*
4.9 Fallas en SOA *101*
4.10 Malas prácticas en SOA *102*
4.11 Desafíos SOA *104*
4.12 Costo - Beneficio. *106*
4.13 Electronic Data Interchange *111*
4.13.1 Ventajas 112
4.13.2 Desventajas 113
4.13.3 EDI y SOA 114
5. METODOLOGÍAS ÁGILES DE DESARROLLO DE SOFTWARE Y SOA 116
5.1 Introducción *116*
5.2 Delivery lifecycle en SOA. *116*
5.2.1 Fase de Análisis 118
5.2.1.1 Objetivos 118
5.2.1.2 Proceso 118
5.2.2 Fase de Diseño 119
5.2.2.1 Objetivos 119
5.2.2.2 Proceso 119
5.2.3 Fase de Desarrollo 124
5.2.4 Fase de Pruebas 124
5.2.5 Fase de Deployment 125
5.2.6 Fase de Administración 125
5.2.7 Estrategias de Delivery 126
5.2.8 Metodologías SOA 128
5.3 Metodologías en detalle. *129*
5.3.1 Ágiles. 129
5.3.2 RUP 130
5.3.3 SOMA 131
5.3.4 XP. 135
5.3.5 SUN Repeatable Quality (RQ). 137
5.3.6 SOUP. 139
5.3.7 Lean Software Development 143
5.3.8 CBDI-SAE. 145
5.3.9 Service-oriented analysis and design (SOAD). 149
5.3.10 Steve Jones' Service Architectures 152
5.3.11 BEA 158
5.3.12 Service Oriented Architecture Framework (SOAF). 165
5.4 Algunos ejemplos *165*
5.4.1 RUP, Agile, Project Driven y Shlaer-Mellor 165
5.5 Comparaciones y análisis exploratorio *171*
5.5.1 Características básicas 171
5.5.2 Encuesta 172
5.5.2.1 Análisis de resultados 175
5.5.3 Conclusiones. 187

6. LINEAMIENTOS RESCATADOS Y VISION A FUTURO 189
7. GLOSARIO 190
8. ANEXOS 193
8.1 Comparación de metodologías de desarrollo en SOA 193
8.2 Mapa de Artefactos RUP. 194
9. BIBLIOGRAFÍA 195

Agradecimientos

Quiero agradecer en primer lugar a toda mi familia, porque me han regalado su gran apoyo e incondicional paciencia durante el desarrollo de éste trabajo; en especial a mis padres, a mis 11 hermanos con sus respectivas familias y a mis 31 adorables sobrinos.

También quiero agradecer a Rodrigo por todo el cariño y la confianza que siempre puso en mí y que siempre fue un gran aliento durante la realización de este trabajo.

A mis amigos y compañeros de estudio por su apoyo, sus experiencias, sus consejos y críticas y por sobre todo por su total acompañamiento.

A mi director de tesis que me brindó todo el acompañamiento necesario, fundamental en el desarrollo de éste trabajo.

A todos aquellos que colaboraron respondiendo la encuesta.

A todas las personas que contribuyeron directa o indirectamente aportando su granito de arena.

ORGANIZACIÓN DEL TRABAJO.

En el capítulo 1 se plantea el objetivo del trabajo. Se indica el por qué de la necesidad de adherirse a una metodología de desarrollo de software, independientemente de la arquitectura o tecnología que involucre un proyecto.

En el capítulo 2 se incluye una breve reseña del surgimiento de las metodologías de desarrollo desde las tradicionales a las ágiles, las híbridas, y por último la aparición de otras aplicables a proyectos que poseen una arquitectura orientada a servicios.

En el capítulo 3 se describen las metodologías de desarrollo tradicionales y las ágiles. Se hace una análisis de sus características, las ventajas y limitaciones que cada una de ellas posee. Para finalizar se efectúan comparaciones entre ellas.

En el capítulo 4 se estudian las arquitecturas orientadas a servicios. Se detallan sus características, conceptos, componentes, ventajas y limitaciones. También se hace un estudio acerca de el Intercambio Electrónico de Datos y la ventaja de SOA como medio de comunicación entre servicios. Por último se hace un pequeño análisis en base a los costos y beneficios involucrados producto del uso de SOA.

En el capítulo 5 se indican las fases presentes en el ciclo de vida de un proyecto SOA y se analizan las metodologías aplicables al desarrollo de software sobre arquitecturas orientadas a servicios. Se analizan con el objetivo de conocer sus características, ventajas y limitaciones. Por último se efectúa una comparación entre ellas.

En el capítulo 6 se mencionan lineamientos y proyecciones a futuro en base a las metodologías de desarrollo sobre proyectos que involucran SOA. Esto se realiza partiendo de las conclusiones efectuadas al final del capítulo 5.

El capítulo 7 posee un glosario con la terminología específica empleada durante el desarrollo de los capítulos anteriores.

El capítulo 8 consiste en anexos referenciados en éste trabajo.

1. INTRODUCCIÓN

Las organizaciones se enfrentan hoy día a mundo cambiante que exige tener la flexibilidad suficiente para adaptarse y responder rápidamente a los cambios, aprovechar oportunidades, plantearse y lograr desafíos, y sobre todo cumplir con las necesidades del negocio y los clientes.

La tecnología de la información desempeña un rol crítico en esta realidad, convirtiéndose en uno de los pilares fundamentales para lograr flexibilidad, rapidez y reducción de costos. Como nuevo paradigma, la orientación a servicios pareciera introducir grandes expectativas, oportunidades y desafíos en este terreno. Algunas organizaciones ya han optado por una Arquitectura Orientada a Servicios, otras en cambio se encuentran evaluando su implementación en pos de obtener beneficios en sus negocios. Ya sea un caso o el otro, es fundamental que cada una conozca acerca de la arquitectura en si, requerimientos, componentes, beneficios, fallas (y cómo evitarlas), buenas prácticas, patrones, casos de éxito, métricas para medir el impacto en el negocio, etc.

Más allá de la arquitectura, la tecnología y los estándares es necesario contar con una metodología que permita el éxito de la adopción e implementación de SOA en una organización. Con este objetivo y con el de dar soporte al desarrollo de software, la historia ha mostrado el surgimiento y evolución de varias metodologías. Ya sea desde el desarrollo en cascada, centrado en la información, donde era necesario completar una fase para poder seguir a la próxima, hasta las ágiles y las híbridas.

En el caso de desarrollos donde se está ante la presencia de SOA, al igual que con cualquier otro tipo proyecto de desarrollo de software, es necesario contar con una metodología que permita entre otras cosas: descubrir las actividades del negocio que necesitan ser soportadas, descubrir servicios, lograr un buen diseño, orquestarlos, satisfacer las necesidades del negocio, asegurar la calidad, adaptarse a los cambios, preparar a la organización de forma tal que pueda proyectar nuevas capacidades, proveer nuevos servicios y reutilizar los existentes.

Considerando a las metodologías como actor indispensable en el desarrollo de software y a SOA un paradigma desafiante para los negocios en la actualidad, el objetivo de este trabajo es comprender las características y componentes en SOA, su ciclo de vida, y aquellas metodologías que aportan soluciones en este escenario.

Debido a que las metodologías que surgieron para dar soporte a SOA se basan en las tradicionales o las ágiles primero se dará un breve detalle de cada una de ellas.

Para finalizar, se efectuará una encuesta y se analizarán los resultados con el fin de conocer cuáles son aquellas metodologías que se adaptan mejor a proyectos con SOA y tienen mejores perspectivas a futuro.

2. ANTECEDENTES

Desde comienzos de los años cincuenta con la aparición de los primeros sistemas de información, han surgido diferentes métodos, paradigmas y modelos de procesos con la intención de soportar y ser capaces de manejar los esfuerzos puestos en el desarrollo de los mismos. Sin embargo, algunos métodos de desarrollo en lugar de lograr flexibilidad y ser más livianos, se tornaron tediosos en cuanto a la documentación y a los procesos que los desarrolladores debían respetar. A éstos modelos, debido a lo indicado anteriormente, se los llamó pesados o métodos tradicionales. Un ejemplo de ellos es el análisis y diseño estructurado.

Con cada implementación de métodos tradicionales, el trabajo comenzaba con la elicitación y documentación de un conjunto completo de requerimientos, seguido de una arquitectura y diseño de alto nivel, desarrollo y luego las pruebas.

Con el paso del tiempo, la evolución rápida de la industria y la tecnología indicaron a las empresas que sus sistemas debían poder adaptarse a ellos. Así fue como se dieron cuenta que, siguiendo bajo las metodologías tradicionales les sería muy complejo y hasta veces imposible cumplir con todos los procedimientos, artefactos y procesos que éstas requerían. Como consecuencia de esto e intentando dar respuesta a éste escenario las consultoras comenzaron a desarrollar métodos y buenas prácticas que les permitiesen poder adaptarse a los cambios. Este fue el inicio de las metodologías ágiles, que se componen de un conjunto de técnicas que comparten principios básicos.

3. METODOLOGÍAS DE DESARROLLO DE SOFTWARE

En este capítulo se describirán, en primer término, las metodologías de desarrollo de software tradicionales, se conocerán sus características, sus fortalezas y debilidades. Luego, con el mismo criterio se estudiarán las metodologías ágiles. Éstas nos servirán como base para estudiar a las metodologías de desarrollo de software que soportan SOA, siendo éstas el objetivo del presente trabajo.

3.1 METODOLOGÍAS TRADICIONALES DE DESARROLLO DE SOFTWARE

Las metodologías que se detallarán en esta sección son: proceso en cascada, desarrollo iterativo, proceso espiral, proceso unificado de desarrollo y por último Jackson Development Methods.

3.1.1 PROCESO CASCADA (Waterfall Process).

Fue introducido por Royce en 1970 y luego, hacia 1981 Bohem hizo una extensión de éste modelo agregando pasos adicionales. Una de las contribuciones de éste modelo fue la creación de la cultura del “pensar” antes de codificar, que en los años ochenta se volvió un estándar dada la ausencia de otros modelos [CASC-VS-ITER].

3.1.1.1 Características

El proceso en cascada se originó como una forma de analizar, diseñar y construir de acuerdo a las necesidades de los usuarios. Esta aproximación en el ciclo de vida del desarrollo de sistemas, describe un método de desarrollo que es lineal y secuencial.

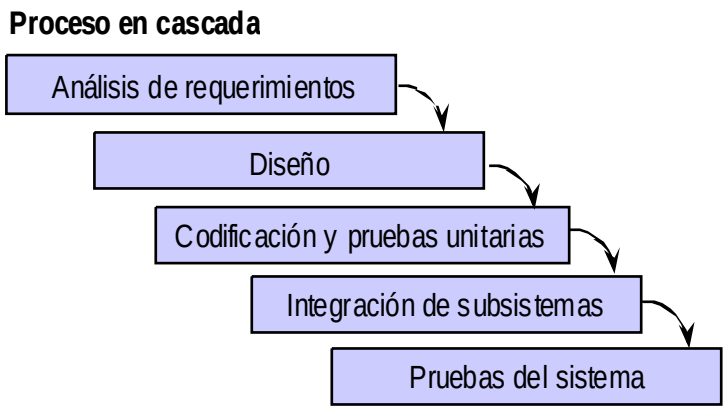
Posee diferentes objetivos en cada fase de desarrollo. Una vez que una fase se completa, se continúa a la siguiente y no hay vuelta atrás.

El comienzo del proceso está dado por un análisis completo de los requerimientos. El objetivo es lograr un conjunto de requerimientos tanto funcionales como no funcionales, definitivos y que sirvan para la siguiente fase. Luego, de la interacción de profesionales (ingenieros, arquitectos, administradores de base de datos, etc.) surge la arquitectura del sistema. A continuación, los desarrolladores llevan a cabo el desarrollo del mismo,

basándose en la documentación antes generada. Por último, el sistema completo es evaluado y puesto en producción.

Es una aproximación guiada por la documentación donde la comunicación recae en la cantidad y calidad de los documentos que se generan en cada fase.

En el diagrama que se muestra a continuación, se indican las fases involucradas en el proceso cascada.



3.1.1.2 Ventajas

La ventaja de éste modelo es que permite la división en departamentos y el control a nivel de proyecto. Permite fijar fechas límites en la planificación para cada etapa de desarrollo y el producto puede ser llevado de etapa en etapa y llevar a cabo el producto “a tiempo”. [OIT-ASD]

Entre sus fortalezas: fue el primer modelo de ingeniería de software, ayudó a desarrollar la cultura del “pensar” antes de codificar y es fácil de entender y adoptar. [CASC-VS-ITER].

3.1.1.3 Limitaciones

Si bien éste proceso parecía efectivo, con la práctica se vio que (1) no se llegaba a cubrir todas las necesidades del usuario, dado que el análisis y la generación de documentación llevaba tanto tiempo que el usuario agregaba nuevos requerimientos y nunca terminaba de estar seguro de lo que quería. Además, lo que veía en producción no terminaba de satisfacer sus necesidades. Asimismo, ocurría que (2) cuando ya se estaba

en la fase de desarrollo surgían cambios y era muy difícil absorberlos y acomodar los tiempos del proyecto. [AGIL-DACS]

En consecuencia, los cambios debían ser congelados para evitar que en medio del desarrollo del proyecto, nuevamente usuarios e ingenieros debieran reunirse, discutir requerimientos, analizar, documentar exhaustivamente, diseñar e implementar el cambio.

Otras limitaciones:

- No permite la revisión. Es decir, una vez que la aplicación está en la fase de prueba, es muy difícil volver hacia atrás y cambiar algo que no fue bien pensado en la fase de análisis de requerimientos. [OIT-ASD]
- Retrasa el manejo de riesgos críticos para el proyecto.
- Mide el avance por la completitud de documentos, esto lleva al síndrome de "producción de papel".
- Retrasa el testing y la integración del sistema.
- No promueve el desarrollo temprano.

3.1.2 PROCESO ITERATIVO.

Surge como mejora al proceso en cascada y se usó de forma exitosa por cuatro décadas en varias organizaciones. Las primeras referencias que se encontraron sobre éste modelo datan del año 1968 reportado por Randell y Zurcher en el centro "IBM T.J Watson Research Center" [IID-BHISTORY].

A menudo es llamado circular o evolutivo.

3.1.2.1 Características

Está compuesto por un conjunto de iteraciones. Cada iteración tiene como objetivo entregar versiones del software y se considera un subproyecto que genera productos de software, y no sólo documentación. De ésta forma permite al usuario tener puntos de verificación y control más rápidos e inducir un proceso continuo de pruebas y de integración desde las primeras iteraciones [ESTB-RUP].

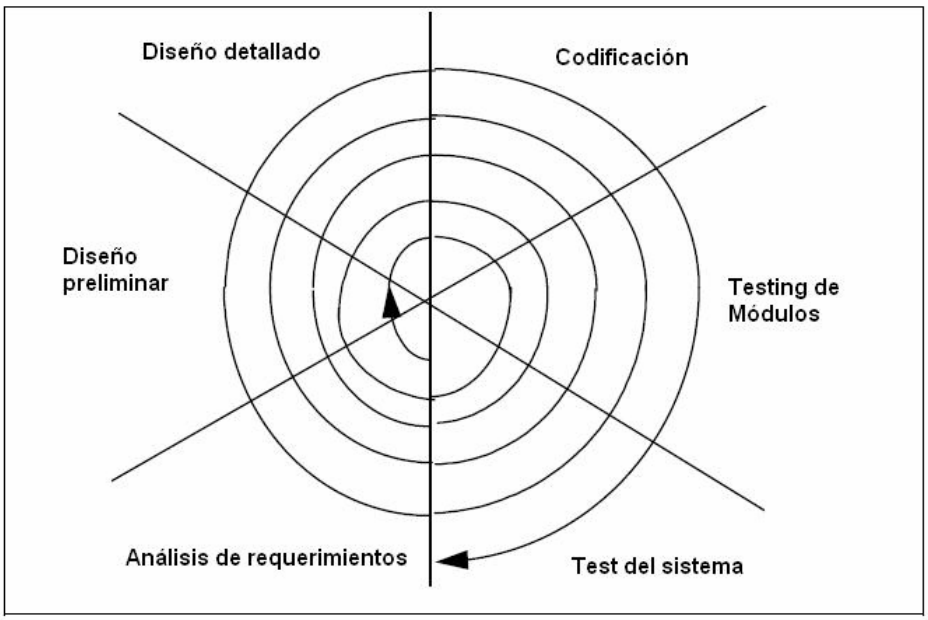
En cada iteración se efectúan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema. El desafío se centra en asegurar que todas las iteraciones sean compatibles [ESTB-RUP].

En 1975, Vic Basili y Joe Turner publicaron un informe que describía cómo este modelo se centraba en el mejoramiento iterativo, es decir, el desarrollo de un sistema de

manera incremental, permitiendo al desarrollador aprovechar lo aprendido en el desarrollo de la versión anterior. Además indicaba que el proceso se comenzaba con un desarrollo sencillo de los requerimientos del sistema y luego se iba mejorando la secuencia evolutiva de versiones hasta completar la totalidad de la funcionalidad requerida [BASILI-TURNER].

Al igual que el proceso en cascada, comienza con una fase de requerimientos, seguida de una fase de diseño y otra de implementación. Luego de esta primera ronda de implementación, se inicia una fase de evaluación con el fin de verificar el éxito o falla del trabajo completado. Para fijar los objetivos de la siguiente ronda de requerimientos, diseño e implementación, se utiliza la devolución de los usuarios (o feedback), tareas de rendimiento, dificultades en codificación, requerimientos pocos claros o inadecuados y herramientas de análisis. Luego de que éstas son completadas, en una segunda instancia, otra ronda de evaluación comienza y se repetirá el proceso hasta que se complete el proyecto [SOA-LIFEC].

A continuación se incluye un diagrama que representa las fases en un método de desarrollo de software iterativo.



En el presente trabajo se describirán otros procesos considerados iterativos, como ser: Modelo Espiral, RUP y XP.

3.1.2.2 Ventajas

La ventaja del uso de éste modelo es que los usuarios finales son involucrados en el proceso de desarrollo. En lugar de esperar hasta que la aplicación sea el producto final, si

bien luego podría no ser fácil efectuar cambios, los problemas son identificados y resueltos en cada fase de desarrollo. [ESTB-RUP]

La construcción completa del sistema, efectuada mediante pequeñas versiones, permitió:

- Que los costos se puedan reducir mediante la definición de pequeños incrementos.
- Que los riesgos puedan ser minimizados debido a la división y a la priorización de las funciones que se consideren centrales.
- Adaptarse mejor a los cambios.
- Alcanzar el diseño de los objetivos de los clientes, quienes no saben cómo definir lo que desean.

Por otra parte el desarrollo iterativo aporta los siguientes beneficios [METH-MARKS]:

- Alcanzar un rápido feedback de los usuarios.
- Incrementar la usabilidad y calidad de la aplicación.
- Descubrimiento temprano de las fallas.
- Incorporar rápidamente nuevas funcionalidades.
- Lograr un equipo más motivado

3.1.2.3 Limitaciones

Eduardo Malaga Chocano en su estudio "Comparative Study of Iterative Prototyping vs. Waterfall Process Applied To Small and Medium Sized Software Projects" [CASC-VS-ITER] indica que bajo ciertas situaciones estudiadas, el modelo iterativo es más efectivo si se trata de incrementar la productividad debido al aprendizaje. Además, permite la finalización temprana y promueve una mejor distribución del tiempo disminuyendo los tiempos libres de los desarrolladores. Por otro lado, el análisis sensible demuestra que el modelo secuencial es más estable en términos de duración y calidad y luego aporta opciones menos riesgosas cuando las condiciones iniciales no son certeras.

Para apoyar el desarrollo de proyectos por medio de este modelo, se han creado frameworks, de los cuales los dos más famosos son Rational Unified Process[1] (RUP) y Dynamic Systems Development Method[2] (DSDM). El desarrollo incremental e iterativo es

[1] http://es.wikipedia.org/wiki/Rational_Unified_Process
[2] www.dsdm.org

también una parte esencial de un tipo de programación conocido como Extreme Programming[3] y los demás frameworks de desarrollo rápido de software.[4]

Se han mencionado como limitaciones el costo extra que significa para una organización, el hecho de que mientras que los usuarios evalúan el software, dejan de ser directamente productivos. Además de lo expresado antes, otra limitación se produce en desarrollos que de antemano se sabe que serán grandes en el consumo de recursos y que insumirán mucho tiempo[5].

Por otra parte, los arquitectos de software todavía necesitan de una base sobre la cual desarrollar sistemas. Esta base se compone de una cierta cantidad de análisis up-front (ver glosario) y de prototipos para construir un modelo de desarrollo en el cual se utilizan patrones específicos de diseño y ERD (Entity Relationship Diagrams, Diagramas de Entidad-Relación o DER en castellano). Se indica que sin estas bases up-front se pueden generar desafíos lo suficientemente significantes en cuanto costos y calidad.

En relación a los costos, también puede tornarse un proyecto muy costoso si: las iteraciones no son lo suficientemente pequeñas para mitigar los riesgos, la posible dificultad para coordinar proyectos de gran envergadura, la tendencia a no documentar el sistema luego de que es completado [METH-MARKS] y la dificultad en predecir exactamente qué funcionalidades están dentro de los tiempos y el presupuesto estimado[6].

3.1.3 PROCESO ESPIRAL

3.1.3.1 Características

Fue desarrollado en 1986 por Boehm. Combina las funcionalidades del modelo en Cascada y de Prototipado pero agregando el análisis de riesgos dentro del proyecto de software. Se propone determinar la viabilidad del proyecto a partir de la identificación de los riesgos, determinar planes para garantizar desde las fases iniciales la eliminación de los errores, donde es menos costoso. Dado que se basa en el modelo de prototipos, permite la entrega, desde las fases iniciales, de versiones del producto ya probados, logrando un proceso continuo de pruebas y retroalimentación.

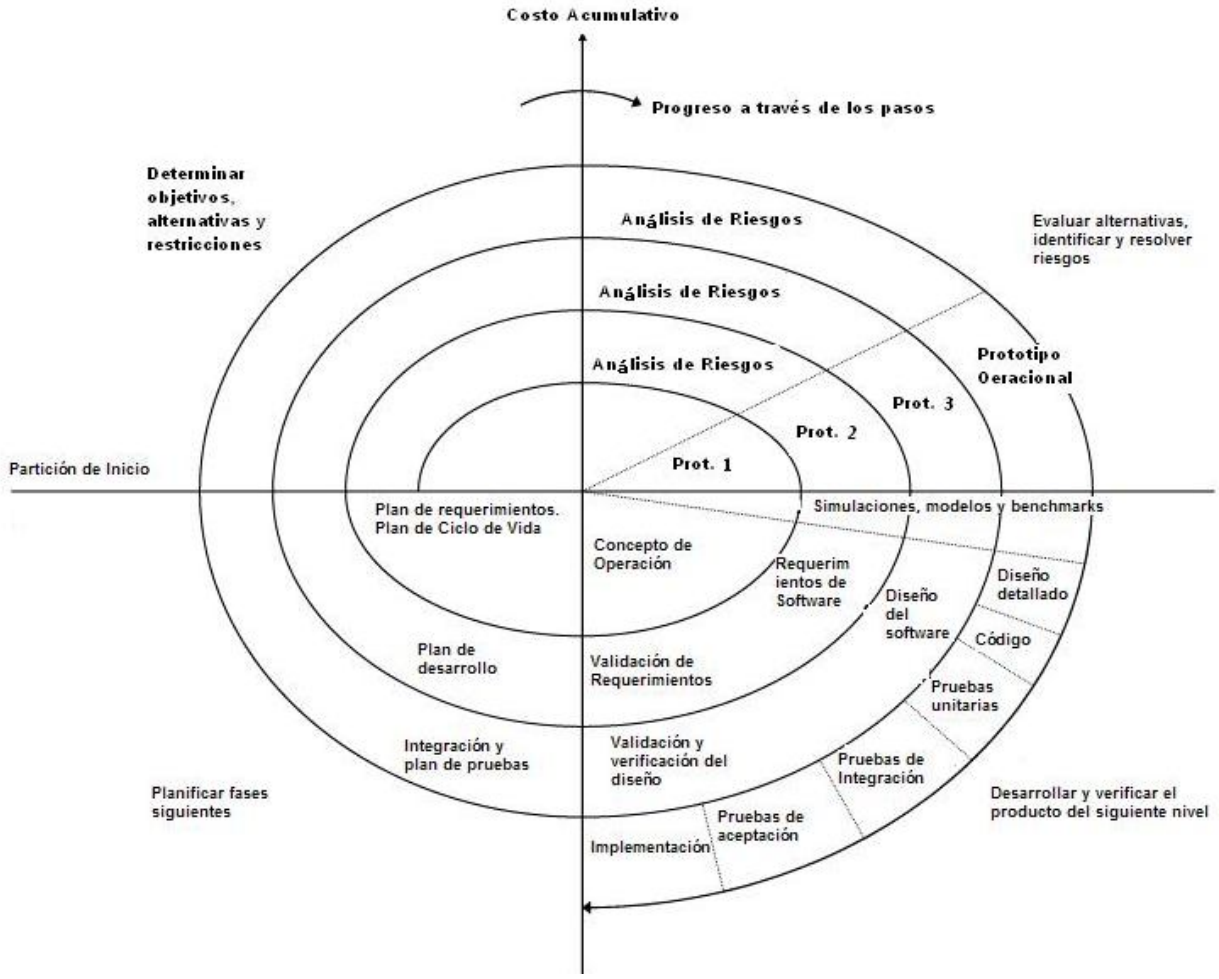
[3] http://www.extremeprogramming.org/, www.xprogramming.com
[4] www.wikipedia.com
[5] http://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente
[6] http://en.wikipedia.org/w/index.php?title=Software_development_process

Enfoca el desarrollo del software a través una serie de ciclos que pueden ser descriptos como un modelo reducido de cascada. El primer ciclo comienza con un delineado de los objetivos, alternativas, restricciones y los riesgos a partir del análisis de los requerimientos. Al finalizar este ciclo generalmente se construyen prototipos con el fin de esclarecer las dudas que hayan quedado en el ciclo y a su vez, el usuario también puede sugerir modificaciones según lo visto en el prototipo.

Los ciclos siguientes se retroalimentan de las etapas anteriores para incrementar el nivel de detalle y el cuidado de los objetivos prospectivos de los sistemas, restricciones y alternativas [MS-RAD].

Al final de cada ciclo, se construye y revisa un prototipo que constituye una versión reducida del sistema en construcción. De esta forma, se logra mejorar el nivel de comunicación entre los usuarios y desarrolladores y reducir el nivel de abstracción.

A continuación se muestra en un diagrama un espiral con los cuatro cuadrantes que definen actividades.



Adaptado de “Software Risk Management” (Boehm,1989)

3.1.3.2 Pasos involucrados en el desarrollo en Espiral.

1) Los requerimientos del sistema son definidos con el mayor detalle posible.
2) Se efectúa el diseño preliminar.
3) Construcción del primer prototipo a partir del diseño preliminar representando las características del producto final.
4) Se desarrolla un segundo prototipo como sigue:
    - Evaluación del prototipo anterior en términos de fortalezas, debilidades y riesgos.
    - Definición de requerimientos del segundo prototipo.
    - Planificación y diseño del nuevo prototipo.
    - Construcción y Pruebas.
5) Según la opinión del cliente, el proyecto puede ser abortado si el riesgo es demasiado grande. Entre ellos pueden estar involucrado los costos, errores de cálculo en los costos de operación, etc.
6) Se evalúa el prototipo de la misma forma que el anterior. En caso de ser necesario se desarrolla uno nuevo.
7) Se iteran hasta que el cliente considere al prototipo como el producto final luego de los refinamientos ya practicados.
8) Se construye el sistema final basado en el prototipo.
9) Se evalúa y prueba el sistema. Se practican rutinas de mantenimiento para prevenir fallas.

Estos pasos fueron obtenidos de [OIT-ASD].

3.1.3.3 Ventajas

El modelo en espiral es indicado para los desarrollos internos de grandes sistemas.

Los beneficios que aporta son:

- Mecanismos de reducción de riesgos.
- Soporta el trabajo en iteraciones y refleja prácticas del mundo real.
- Es una aproximación sistemática.
- Emplea la metodología de prototipado hasta llegar al producto final.

3.1.3.4 Limitaciones

Un problema que presenta este modelo es la falta de una guía para incrementar el nivel de detalle y precisión luego de cada ciclo.

- Requiere experiencia en la evaluación y reducción de riesgos.
- Es complejo, relativamente dificultoso de seguir estrictamente.
- Aplicable solo en grandes sistemas.

3.1.4 UP - RUP

Antes de describir Proceso Unificado Rational (Rational Unified Process en Inglés ó RUP) se definirá lo que es un proceso.

Un proceso es la definición de un conjunto de actividades y no la ejecución de las mismas.

Un proceso de desarrollo de software consiste en una serie de requisitos, la ejecución de actividades y como producto de ello, el producto de software (siendo un conjunto de artefactos).

Requisitos → Actividades → Producto de Software

Los inicios de RUP se remontan al modelo espiral original de Barry Boehm. Ken Hartman, uno de los contribuidores claves de RUP, colaboró con Boehm en la investigación. Hacia 1995 Rational Software es comprada por una compañía de origen sueco llamada Objectory AB. Rational Unified Process fue el resultado de una convergencia de Rational Approach y Objectory, proceso desarrollado por el fundador de Objectory Ivan Jacobson. Como resultado de esta fusión surgió Rational Objectory Process, siendo la primera versión de RUP que fue puesta en el mercado en 1998, cuyo arquitecto en jefe fue Philippe Kruchten[7].

Es el proceso no solo para la implementación del análisis orientado a objetos, diseño y metodología de programación, sino que también es para modelado de procesos de negocio. Es dirigido por casos de uso, basado en componentes, con centro en el manejo de proyectos, y con énfasis en Unified Modeling Language (Lenguaje Unificado de Modelado - UML) como elección de modelado.

Se dice que es **basado en componentes** dado que la creación de sistemas intensivos en software requiere dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica en un

[7] http://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational

proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o se desarrollan sus componentes [RUP-VAL1].

Provee una aproximación disciplinada para la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad para satisfacer las necesidades de los usuarios finales dentro del tiempo y presupuesto planificados. [RUP-BESTP]

Finalmente, UP es RUP sin los productos copyright de Rational Unified Process (RUP).

3.1.4.1 Características

RUP es un modelo basado en el **modelo en espiral** y teniendo un **proceso iterativo e incremental** de forma que el trabajo pueda dividirse en pequeñas partes o proyectos, logrando un equilibrio entre casos de uso y arquitectura. Cada parte puede verse como una iteración de la cual se obtiene un incremento que produce un crecimiento en el producto. [RUP-VAL1]

Para cada iteración existe una planificación, un análisis y algunas actividades específicas.

Una iteración puede realizarse por medio de una cascada pasando por los flujos fundamentales (Requisitos, Análisis, Diseño, Implementación y Pruebas). También existe una planificación de la iteración, un análisis de la iteración y actividades específicas. Al finalizar se realiza una integración de los resultados con lo obtenido de las iteraciones anteriores, se pueden descubrir nuevos requerimientos o verificar si hubo cambios en alguno de forma tal que requiera modificaciones en la iteración posterior [RUP-VAL1].

Organiza las iteraciones en etapas y fases proponiéndose obtener una estructura sólida y ajustable a las necesidades particulares de cada organización y es por ello que se lo denomina **Proceso Configurable**.

Su objetivo es poder ajustarse tanto a proyectos grandes como a los pequeños, brindando un kit de desarrollo que permite la configuración de procesos.

RUP Incrementa la productividad del equipo brindando a cada miembro del equipo un fácil acceso a la base de conocimiento con líneas guía, plantillas y mentores de herramientas para todas las actividades críticas de desarrollo. El tener acceso a la misma base de conocimiento, asegura a todos los integrantes del equipo el compartir un lenguaje común, proceso y vista de cómo desarrollar software [RUP-BESTP].

Las actividades de RUP crean y mantienen modelos. En lugar de enfocarse en la producción de gran cantidad de documentos, enfatiza el desarrollo y mantenimiento de

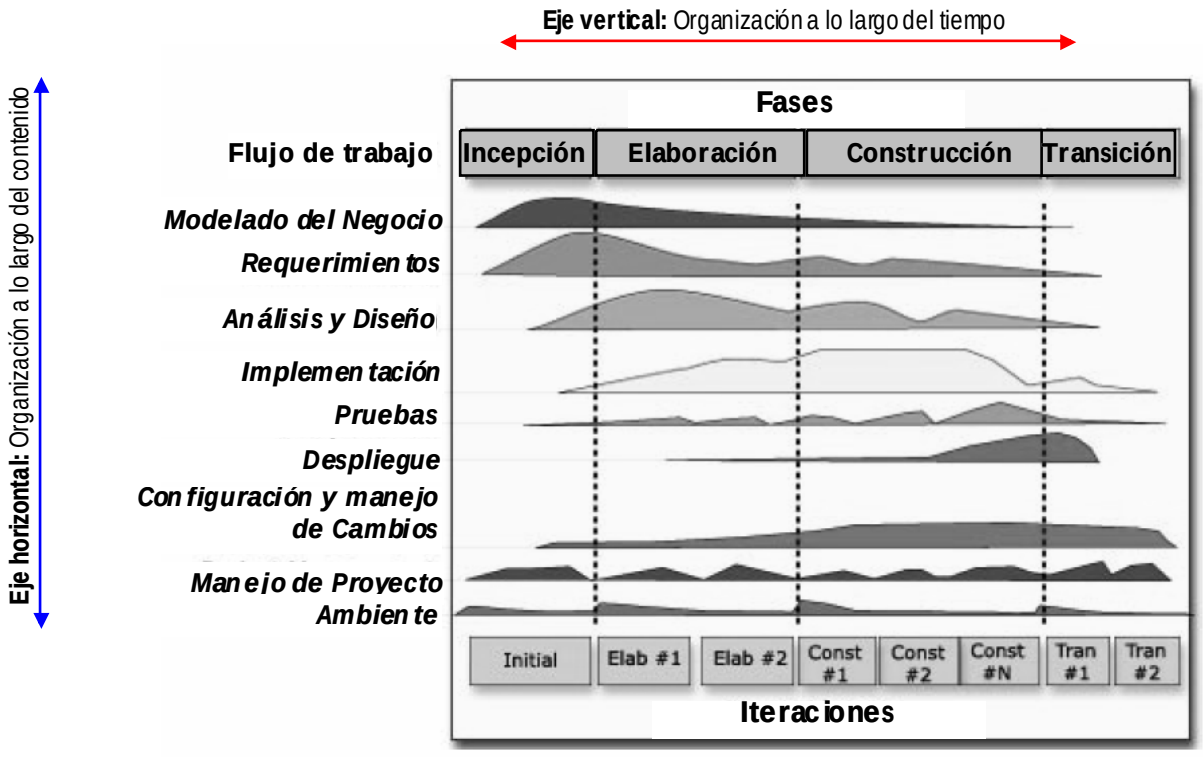
modelos siendo éstos representaciones semánticamente ricas del sistema que se está desarrollando.

El ciclo de vida del software es dividido en ciclos. RUP divide un ciclo de desarrollo en cuatro fases: Incepción, Elaboración, Construcción y Transición (Ver Figura "Las dos dimensiones de RUP"). Cada fase finaliza con un hito bien definido, es decir un punto en el tiempo en que se deben tomar decisiones críticas y luego cumplir los objetivos propuestos.

El proceso se expresa en dos dimensiones. La primera (horizontal) representa el aspecto dinámico del proceso expresado en términos de ciclos, fases, iteraciones e hitos. El producto de software es diseñado y construido en una sucesión de iteraciones incrementales. Esto permite que se efectúen pruebas y validaciones de ideas de diseño tanto como la mitigación de riesgos de forma temprana en el ciclo de vida. La segunda (vertical) representa el aspecto estático del proceso descrito en términos de componentes del proceso: actividades, disciplinas, artefactos y roles [RUP-PKR].

En el eje horizontal, las disciplinas agrupan actividades relacionadas lógicamente y en el eje vertical, en una iteración se atraviesan todas las disciplinas.

Las dos dimensiones de RUP quedan representadas como se muestra debajo. A continuación se dará una breve descripción de la esencia de cada eje. Para comenzar, se indicarán las bases de cada fase y en segundo término se comentará el flujo de trabajo (Workflow en inglés).



Las dos dimensiones de RUP

3.1.4.2 Fases

Incepción

Se establece el caso de negocio para el sistema y se delimita el alcance del proyecto. Para ello se identifican todas las entidades externas con las que el sistema interactuará y se define la naturaleza de esta interacción en un alto nivel. Se identifican los casos de uso más significativos. El caso de negocio incluye criterios de éxito, los riesgos, estimación de recursos necesarios y un plan que muestre las fechas de los hitos más importantes [RUP-BESTP].

El desarrollo de los casos de uso más significativos implica explorar un pequeño conjunto de requerimientos, quizás un 10%, con el fin de obtener un orden de sentido de magnitud del alcance, los riesgos clave, y decidir cuándo comenzar la fase de elaboración [RUP-FAIL].

Luego se valida que el proyecto cumpla con el siguiente criterio de evaluación [RUP-PGUIDE]:

- Concurrencia de los actores del proyecto en la definición del alcance y estimación de costos y tiempos, que serán refinados en fases posteriores.
- Comprensión de los requerimientos como evidencia para la fidelidad de los casos de uso primarios.
- Credibilidad de las estimaciones de costo y tiempos, prioridades, riesgos, y desarrollo del proceso.
- Profundidad y amplitud de cualquier prototipo de arquitectura que fue desarrollado.
- Entregas realizadas contra las planeadas.

El hito a cumplir en la finalización de esta fase es el ciclo de vida de los objetivos.

Si el proyecto no pasa este hito (llamado Hito Objetivo del Ciclo de Vida), puede ser cancelado o repetirse esta fase luego de ser rediseñado con un mejor criterio.

Elaboración

El objetivo de ésta fase es analizar el dominio de problema, establecer la arquitectura, desarrollar el plan de proyecto, demostrar que la arquitectura propuesta soportará la visión bajo costos y tiempos razonables y por último eliminar los elementos de alto riesgo del proyecto.

Las decisiones sobre arquitectura deben ser tomadas en base al conocimiento de todo el sistema: su alcance, principales requerimientos funcionales y no funcionales como son los de rendimiento [RUP-BESTP].

Construir la arquitectura significa programar, integrar y probarla. Para esto, será necesario explorar en mayor detalle los requerimientos, quizás el 80%, mientras que en

paralelo se implementan las partes centrales más riesgosas del sistema. Los requerimientos pueden cambiar significativamente durante esta fase, por medio ciclos, en respuesta a las evaluaciones sobre implementaciones parciales. En contraste con la definición de los requerimientos en el proceso en cascada, la mayoría de los requerimientos son definidos en paralelo al desarrollo de la arquitectura base, e informados desde la retroalimentación de éste desarrollo actual [RUP-FAIL].

La construcción de un prototipo se hará en una o más iteraciones, dependiendo del alcance, el tamaño, riesgos del proyecto. Si bien el esfuerzo se centra en la elaboración de los casos de uso identificados como críticos en la fase de incepción, no se excluye el desarrollo de uno o más prototipos para mitigar riesgos específicos.

Según Scott Ambler [RUP-MANAG] se debe cumplir:

- La visión del proyecto es estable y realista.
- Los requerimientos para el proyecto.
- La arquitectura es estable y satisface los requerimientos.
- Los riesgos seguirán siendo manejados.
- Los gastos son aceptables y las estimaciones razonables han sido creadas para futuros costos y planificaciones.
- El equipo del proyecto tiene chances de triunfar.
- Tanto el plan de iteración detallado para las siguientes iteraciones de Construcción, como el plan de proyecto de alto nivel están definidos.

El hito a cumplir en la finalización de esta fase es el ciclo de vida de la arquitectura.

Construcción

En esta fase se construye el producto por medio de una serie de iteraciones en las cuales se selecciona un grupo de casos de uso, se refina el análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo y se itera hasta que se termine la implementación del producto. Se deben conseguir versiones funcionales (alfa, beta, y otras de prueba) no bien sea posible. A su vez, todos los componentes, características y requisitos deben ser implementados, integrados y probados en su totalidad, obteniendo una versión aceptable del producto [RUP-VAL1].

Se desarrollan todos los elementos pendientes de la fase de Elaboración, se efectúa la integración y el aseguramiento de calidad, y se prepara todo para el deploy (deploy o deployment puede ser instalación en producción). En esta fase, los requerimientos cambian en menor medida [RUP-FAIL].

Según Scott Ambler [RUP-MANAG] al finalizar los usuarios deben aceptar que:

- El software y la documentación son aceptables.

- Los interesados y el negocio están listos para que el sistema sea desplegado.
- Los riesgos se manejan de forma efectiva.
- Los gastos son aceptables y las estimaciones razonables han sido hechas para futuros costos y planificaciones.
- Los planes de iteración para la siguiente fase de Transición tanto como el plan de proyecto de alto nivel están posicionados.

El hito que marca la culminación de esta fase es la Capacidad Operacional Inicial (COI).

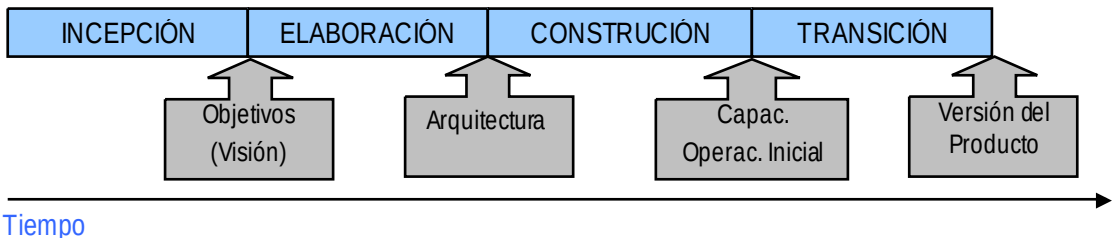
Transición

En esta fase todos los esfuerzos se centran en el pasaje del sistema a producción y en manos de la comunidad del usuario. Las personas encargadas de las pruebas y los usuarios finales ejecutarán pruebas sobre el sistema encontrando puntos que pueden luego ser reconstruidos o puestos a punto. Se efectúa el entrenamiento de los usuarios finales, operadores y personal de soporte [RUP-MANAG].

Una vez que el producto está puesto en producción, normalmente se requiere entrenar al usuario en el manejo del mismo, el desarrollo de nuevas versiones, la corrección de algunos problemas o la finalización de funcionalidades que fueron antes pospuestas [RUP-BESTP].

El hito que marca la culminación de esta fase es la entrega de la versión del producto.

La imagen que sigue muestra de forma gráfica los hitos en la finalización de cada una de las fases antes mencionadas. Cada hito representa un punto en el tiempo definiendo la toma de decisiones críticas y el logro de las metas claves para la siguiente fase.



A continuación se incluye una tabla en la que se detallan **los artefactos básicos** presentes en cada fase del proceso de desarrollo de RUP. Ver Anexo 8.2 con el Mapa de Artefactos.

Fase	Artefactos
Incepción	▪ Documento de visión general del los requerimientos centrales del proyecto, principales funcionalidades y restricciones. ▪ Modelo inicial de caso de uso, entre el 10 y 20% completado. ▪ Glosario inicial del proyecto. ▪ Caso de uso ▪ Caso de negocio inicial, que induya el contexto, criterios de éxito y pronóstico financiero. ▪ Estimación inicial de riesgos. ▪ Plan de proyecto mostrando fases e iteraciones. ▪ Modelo de negocio si fuere necesario. ▪ Prototipos.
Elaboración	▪ Modelo de casos de uso (80% completado), siendo todos los usuarios identificados. ▪ Requerimientos suplementarios capturando requerimientos no funcionales y otros no asociados con un caso de uso específico. ▪ Descripción de Arquitectura de Software. ▪ Prototipo de arquitectura ejecutable. ▪ Lista de riesgos y casos de negocio revisados. ▪ Un plan de desarrollo para todo el proyecto que muestre las iteraciones y criterio de evaluación para cada iteración. ▪ Un caso actualizado de desarrollo especificando el proceso a ser usado. ▪ Manual de usuario preliminar (opcional).
Construcción	▪ El producto de software integrado en la plataforma adecuada. ▪ Manual de usuario. ▪ Descripción de la versión actual.
Transición	▪ Pruebas para validar el nuevo sistema contra las expectativas del usuario. ▪ Operación paralela con los sistemas legacy (ver Glosario) que están siendo reemplazados. ▪ Conversión de bases de datos operacionales. ▪ Entrenamiento de usuarios y administradores. ▪ Distribución y venta de productos de marketing.

3.1.4.3 Flujos de trabajo

RUP define cuatro elementos que serán involucrados en los flujos de trabajo. Los mismos son:

- Los **roles** que define el comportamiento y responsabilidades de un individuo y que responden a la pregunta ¿Quién?.
- Las **actividades** que responden a la pregunta ¿Cómo? y representan una unidad de trabajo que una persona con un rol puede realizar.

- Los productos ó **artefactos**, que responden a la pregunta ¿Qué? y representan un trozo de información que es producido, modificado o usado durante el proceso de desarrollo de software.
- Los **flujos de trabajo** de las disciplinas que responde a la pregunta ¿Cuándo? y son secuencias de actividades que producen un resultado de valor observable. En términos de UML puede ser expresado con un diagrama de secuencia, de colaboración o de actividades [RUP-BESTP].

Los flujos de trabajo definidos por RUP son:

- **Modelado de negocio**, cuyo objetivo es comprender y modelar la organización donde se desarrollará el producto.
- **Requisitos**, que especifican qué tiene que cumplir el sistema que se construya y a su vez ofician de contrato a cumplir.

Se produce el *Modelo de Casos de Uso* que es la especificación completa de todas las formas posibles de utilizar el sistema.

- **Análisis y diseño**, que tienen como principal objetivo traducir los requisitos a una especificación que describe cómo implementar el sistema.

El análisis consiste en tener una visión del sistema donde se vea qué es lo que el sistema hace (requisitos funcionales).

Se produce el *Modelo de Análisis*, donde en cada iteración se elige un conjunto de casos de uso a reflejar en él. Se identifica y describe los Casos de Uso para cada iteración, se lee la descripción de cada uno y se elaboran las realizaciones de los casos de uso.

El diseño es un refinamiento del análisis que tiene en cuenta el cómo el sistema cumple los objetivos (requisitos no funcionales) [RUP-BESTP].

Se toma el Modelo de Análisis como entrada principal y se adapta al entorno de implementación elegido. Este modelo es la especificación de la implementación, los casos de uso son implementados en términos de las clases de diseño.

- **Implementación.** Se realiza el Modelo de Implementación. Se implementan las clases y objetos en archivos fuente, binarios y/o ejecutables que conforman el sistema ejecutable. Se realizan además, pruebas sobre subsistemas, se corrigen errores encontrados, se sigue un plan de integración para definir qué subsistemas y en qué orden se implementarán.
- **Pruebas**, con el fin de evaluar que se cumpla con lo especificado en los requerimientos (funcionales y no funcionales) y con la calidad del producto. Para ello se crea un Plan de Pruebas compuesto por casos de prueba y procedimiento de prueba. Mediante la identificación temprana de los casos de uso, podemos comenzar pronto la planificación de las actividades de prueba.

- **Despliegue** (deploy en inglés) que consiste probar, empaquetar, distribuir, instalar el producto. Capacitar, proveer asistencia a los usuarios, etc.
- **Gestión del proyecto**, balancea la gestión de objetivos, riesgos y restricciones para desarrollar un producto que respete los requisitos definidos. Los objetivos son: proveer un marco de trabajo para la gestión de proyectos, proveer guías prácticas, planificar, contratar personal, ejecutar y monitorear el proyecto y proveer un marco de trabajo para gestionar los riesgos [RUP-BESTP].
- **Configuración y control de cambios**, mantener la integridad de los artefactos y mantener la información acerca del proceso evolutivo que se ha seguido.
- **Entorno**, tiene como objetivos: la selección y adquisición de herramientas, establecer y configurar las mismas para que se ajusten a la organización, configurar y mejorar tanto el proceso como los servicios técnicos [RUP-BESTP].

3.1.4.4 Buenas Prácticas.

Para lograr un equilibrio entre la entrega de software de calidad y la entrega rápida es esencial entender cuáles son los elementos esenciales del proceso y lograr una adaptación del proceso para que se ajuste mejor a las necesidades del proyecto. Muchos especialistas ya han incursionado en este tema y han aportado buenas prácticas. RUP implementa buenas prácticas de otras aproximaciones. Las mismas se indican a continuación.

Desarrollo iterativo.

Se requiere un proceso de desarrollo iterativo que permita incrementar la comprensión del problema a través de sucesivos refinamientos, y desarrollar de forma incremental una solución efectiva en múltiples iteraciones. RUP soporta el desarrollo iterativo, permite localizar los mayores riesgos en cada etapa en el ciclo de vida y reducirlos de forma significativa en el proyecto. Esto se lleva a cabo mediante la entrega de versiones ejecutables que permiten involucrar constantemente al usuario y obtener un feedback de él. Los equipos de desarrolladores se mantienen enfocados en la producción de resultados, y las verificaciones de estado efectuadas frecuentemente ayudan a que el proyecto se mantenga dentro de los plazos pactados. A su vez, facilita la absorción de cambios en requerimientos, funcionalidades o planificaciones. [RUP-BESTP]

En resumen, y de acuerdo con Philippe Kruchten los beneficios de una aproximación iterativa son [RUP-INTRO]:

- Los cambios son más manejables.
- Los riesgos se pueden mitigar con anterioridad.

- Hay un alto nivel de reuso.
- El equipo del proyecto puede aprender durante el proceso.
- Los productos son mejores y de mayor calidad.

Administración de requisitos.

Describe cómo elicitar, organizar y documentar la funcionalidad requerida y las restricciones; hacer el seguimiento y documentación de las decisiones; capturar y documentar requerimientos de negocio.

Los beneficios de un efectivo manejo de los requerimientos son: mejorar el control de proyectos complejos; mejorar la calidad del software y la satisfacción de cliente; reducir los costos y demoras y mejorar la comunicación del equipo de trabajo [RUP-INTRO].

Arquitectura basada en componentes.

El proceso se enfoca en el desarrollo temprano de la línea base de una arquitectura ejecutable robusta antes de entregar todos los recursos para el desarrollo en escala. Describe cómo diseñar una arquitectura flexible, adaptable a los cambios, entendible de forma intuitiva, y que promueva en forma efectiva el reuso de componentes. RUP soporta el desarrollo basado en componentes, siendo éstos módulos no triviales, subsistemas que completan una función clara y que luego son ensamblados en una arquitectura bien definida [RUP-BESTP].

Se centra entonces en la identificación temprana de los componentes necesarios, y luego, las pruebas se organizan en torno a éstos componentes.

Modelo Visual (UML).

El proceso unificado es una guía para el uso efectivo de Lenguaje Unificado de Modelado. Describe cuáles son los modelos necesarios, el porqué de ellos y cómo se construyen.

Gran parte de RUP se enfoca en el desarrollo y mantenimiento de modelos para entender y mostrar de forma simplificada sistemas complejos que no pueden ser comprendidos de forma completa de otra manera [RUP-INTRO]. Por ello hace uso de un lenguaje de modelado que brinda un medio estándar de escribir el esquema de un sistema cubriendo ítems conceptuales como son los procesos de negocio y las funciones del sistema tanto como ítems concretos con clases escritas en un lenguaje de programación específico, esquemas de base de datos, y componentes de software reusable [UML-BOOCH].

Verificación continua de calidad.

La calidad debe revisarse teniendo en cuenta los requerimientos basados en la seguridad, funcionalidad y desempeño tanto de la aplicación como del sistema. RUP asiste el planeamiento, diseño e implementación, la ejecución de las pruebas su posterior revisión. El aseguramiento de la calidad es construido dentro del proceso, en todas las actividades críticas, involucrando a todos los participantes, usando medidas y criterios objetivos y no tratados como una actividad separada y realizada sobre un grupo separado [RUP-BESTP].

Administración de cambio.

RUP se propone la mejora constante del proceso de desarrollo de software.

Si se hace foco en las necesidades de desarrollo de una organización, el manejo de los cambios es una aproximación sistemática para administrar los cambios en los requerimientos, diseño e implementación. A su vez cubre actividades importantes de seguimiento de defectos, malos entendidos así como la asociación de esas actividades con artefactos y versiones específicas. El manejo de los cambios está atado a la configuración del manejo y a sus mediciones [RUP-INTRO].

3.1.4.5 Ventajas.

A su vez, en [RUP-ELECT] se definen las ventajas y desventajas teniendo en cuenta determinadas características de RUP. A continuación se mencionan las ventajas.

Teniendo en cuenta que RUP es una metodología:

- Con aproximación iterativa: Los riesgos son tomados en consideración de forma temprana, los cambios son más manejables, existe un alto nivel de reuso, el equipo del proyecto aprende con la experiencia a lo largo del proceso y existe revisión continua de calidad.
- Con centro en la arquitectura: gana control intelectual y oportunidad para el reuso.
- Orientada a Casos de Uso: expresada desde la perspectiva de usuarios, escrita en lenguaje natural y por lo tanto de fácil comprensión, alto grado de trazabilidad, provee una forma simple de descomponer los requerimientos en partes, define casos de prueba y posee planificación de iteraciones.

3.1.4.6 Limitaciones.

A pesar de las ventajas mencionadas anteriormente, RUP como un todo puede ser difícil de entender y de implementar de forma apropiada si el equipo involucrado en el proyecto no posee la experiencia suficiente en el control y planeamiento de proyectos.

Las limitaciones encontradas son:

- En cuanto al tamaño, es extensivo y de difícil comprensión.
- Alta complejidad y gran amplitud que requiere su personalización para ajustarse a una determinada compañía antes de poder usarse.
- Requiere grandes esfuerzos iniciales e inversión para comenzar su uso.
- El manejo ad hoc de requerimientos.
- Comunicación imprecisa e incomprensible.
- Inconsistencia en requerimientos, diseño e implementación.
- Falta de pruebas.
- Estado de juicio subjetivo.
- Inhabilidad para manejar riesgos.
- Cambios sin control.
- Automatización de desarrollo ineficiente.

Esto se acentúa más en proyectos donde intervienen pocas personas, alrededor de 5 por ejemplo. En otros casos, como en las compañías medianas o grandes, es apropiado en el sentido que permite ser adaptado fácilmente a la estructura de la organización y abarcar todos los aspectos involucrados en el desarrollo del software, en la planificación y control, en el aseguramiento de la calidad, reuso de componentes, arquitectura, etc.

A continuación se incluyen siete pasos publicados que pueden llevar a la ruina la implementación de RUP en las organizaciones, los mismos fueron extraídos de [RUP-FAIL].

1. Sobre-imponer el pensamiento en cascada.

Una forma de sobre-imponer el pensamiento en cascada sobre RUP es considerar que en la fase de incepción se debe realizar la mayor cantidad de los requerimientos, en elaboración hacer el diseño detallado y el modelado, en construcción implementar y en transición la integración, pruebas del sistema y el despliegue.

- *Definir iteraciones muy cortas o muy largas.* La esencia del desarrollo iterativo y de RUP es tomar pequeños pasos como objetivo para una posible implementación imperfecta, hacer la integración rápidamente, el control de calidad, ejecutar las pruebas, obtener rápidamente feedback sobre ello y luego adaptar los requerimientos, el diseño e implementaciones. La clave para no fallar es *"Pasos cortos, feedback, y luego, adaptación"*.

Hay que tener en cuenta que, si se definen iteraciones cortas (por ejemplo dos semanas) y el equipo de desarrolladores es grande (300 por ejemplo), el proyecto no podrá desarrollarse de manera deseada debido a la sobrecarga que esto generaría. Por ello es recomendable que las iteraciones sean de aproximadamente de ocho a diez semanas para equipos grandes.

Por otra parte, un equipo pequeño trabajando de forma temporaria e independiente puede dividir las iteraciones de un proyecto de nivel macro de ocho semanas en iteraciones pequeñas de dos semanas cada una.

- *Refinar los requerimientos antes del diseño*. El proceso en cascada espera lograr la definición y el posterior refinamiento de la mayor cantidad de requerimientos. Esto llevará un tiempo considerablemente grande y posiblemente suceda que cuando se llegue a la implementación de la solución quizás el usuario haya cambiado de parecer y quiera algo diferente a lo que se implementó.

No es tarea sencilla balancear lo que el usuario necesita, lo que realmente desea y elegir lo que es correcto, por eso es recomendable probar con diferentes aproximaciones y ver cuál es la que mejor funciona. Esto es lo que permite un proceso iterativo en contraste con uno en cascada que fuerza la elección de la mejor solución en el primer intento, siendo que esto va en contra de la naturaleza humana y sobre todo de un proyecto de software.

- *Esperar estimaciones creíbles y planes detallados en el inicio del proyecto.* Solo porque se pueda crear un plan que nos muestre cómo marchan las cosas, y los hitos, no significa que realmente lo podamos realizar. Es casi imposible poner fechas precisas en un plan en ausencia de cualquier información real acerca de la cantidad de trabajo a ser realizado.

Las aproximaciones iterativas permiten que con el correr de las iteraciones se tenga más información acerca de los requerimientos reales, una mejor vista de los riesgos reales, y un mejor sentido de las habilidades en el conocimiento de los desafíos.

Tomar decisiones en ausencia de información es lo más riesgoso en un proyecto, por eso, si no se tiene experiencia en proyectos del tipo del que se está requiriendo participación, lo ideal es estirar las estimaciones y contratar gente experimentada en el dominio del problema.

2. Aplicar RUP como un proceso pesado y predictivo.

Una de las formas en que RUP puede fallar es aplicando prácticas de procesos pesados:

- Planear la totalidad del proyecto iterativo en detalle y de forma temprana definir cantidad y fechas de todas las iteraciones y también especificar lo que pasará en ellas.

- Crear la mayor cantidad de artefactos posibles.
- Agregar gran formalidad a los procesos.

Para lograr un proceso más ágil y liviano los autores de RUP pensaron que mejor sería:

- Crear un conjunto mínimo de actividades y de artefactos RUP que agreguen valor.
- Evitar planes detallados para todas las iteraciones y en su lugar armar planes de alto nivel que estimen el fin del proyecto y los mayores hitos pero sin detallar cómo llegar a ellos. También se puede pensar en otro plan de iteración donde solo se planee a grandes rasgos la iteración.

3. No contar con personal capacitado en la tecnología de Objetos.

Dado que RUP está enfocado en el desarrollo de sistemas con orientación a objetos, el no tener desarrolladores calificados en esto será un factor importante que podría marcar el fracaso del proyecto.

4. Quitar valor al desarrollo iterativo adaptativo.

Una de las características más importantes de RUP es el desarrollo iterativo. Hay diversas formas de ignorar las implicancias de un desarrollo iterativo:

- *Adoptar actitudes rígidas y predictivas* ya sea intentando congelar los requerimientos y el diseño en lugar de optar por un cambio y además planear todas las iteraciones futuras en lugar de ir ajustándolas de forma adaptativa.

Teniendo en cuenta que en la transición a un ciclo de vida iterativo los cambios afectan tanto en el nivel organizacional como en el de proyecto, en el primero se debe tomar el cambio como una nueva forma de hacer las cosas y en el segundo, se debe permitir el feedback y el aprendizaje continuo.

- *Prevenir las prácticas del desarrollo iterativo.*

Una aproximación iterativa es la indicada en escenarios donde desde cada rol y a partir de información que quizás no es certera, se estima cómo serán las cosas y qué solución se dará a los problemas. Esto es así dado que ésta aproximación permite que la información sea recogida mientras pueda ser usada para mejorar los planes permitiendo luego cambiar de dirección.

- *No educar a los actores en las implicancias del desarrollo iterativo*. Para que un desarrollo iterativo funcione, los clientes deben estar involucrados de forma que sirvan de soporte a los desarrolladores para que éstos conozcan acerca del negocio y luego puedan resolver problemas. Los clientes deben apreciar la necesidad de su rol en la definición de

qué es necesario desarrollar. Para esto, lo ideal es que se pueda comenzar con el entendimiento del problema que necesita resolución y luego explorarlo de forma evolutiva, trabajando en conjunto desarrolladores y clientes. Esta forma de trabajo permite tener un feedback, esencial en el desarrollo de un proyecto.

- *Modelar por demás y crear artefactos sin valor.* La idea de un desarrollo iterativo es comenzar rápidamente la programación cuando esté disponible una parte de los requerimientos y diseñar, con el fin de obtener feedback.

Si bien se dice que RUP es demasiado extenso, la cuestión esencial radica en discernir qué es lo que se necesita y en base a ello emplear los artefactos necesarios. Para que esto suceda deben surgir cuestionamientos del tipo: "¿Existe el riesgo de que los desarrolladores no entiendan lo que el sistema hace?" Entonces, se necesitará manejo de requerimientos. "¿Hay comportamientos que poseen un flujo complejo de control o son de difícil comprensión? Entonces, se usarán "casos de uso".

5. Evitar mentores que entiendan del desarrollo iterativo.

En el inicio de un proyecto con RUP, es recomendable contar con personal calificado. Un equipo que tenga experiencia en desarrollo adaptativo iterativo, un equipo que posea la experiencia en la tecnología elegida para realizar el proyecto, contar con un mentor que conozca del desarrollo iterativo y tenga influencia en el grupo de líderes del proyecto.

6. Adoptar RUP en un Big Bang.

En la adopción RUP es recomendable planificar y llevar a cabo la introducción cumpliendo dichos planes. Realizar capacitaciones previas y de corta duración con todo el personal. No dejar que pase mucho tiempo entre la capacitación y la puesta en marcha de RUP. Realizar pruebas piloto con un mentor calificado e implementando solo un conjunto mínimo de prácticas RUP de forma que éstas sirvan como experiencia y aprendizaje para la siguiente.

7. Tomar consejos de fuentes mal informadas.

Escoger de forma apropiada las fuentes de información con el propósito de caer en malas interpretaciones.

Por su parte, [RUP-FAIL] definió una lista donde se mencionan las mayores fallas de interpretación de la metodología RUP y que también llevan al fracaso su implementación.

- Pensar que incepción es equivalente a requerimientos, elaboración a diseño y construcción a implementación.
- Pensar que el propósito de la fase de elaboración es para definir modelos de forma completa y cuidadosa, las cuales se traducen en código durante la fase de construcción.
- Pensar que solo los prototipos son creados en fase de elaboración. En realidad, la calidad de producción central de elementos de arquitectura riesgosos deben ser programados en fase de elaboración.
- Intentar definir la mayor cantidad de requerimientos antes de comenzar el diseño y la implementación.
- Intentar definir lo que más se pueda el diseño antes de comenzar la implementación.
- Un gran tiempo se pasa definiendo los requerimientos o el diseño antes de que comience la programación.
- Una organización considera que la duración acorde de una iteración es medida en meses, en lugar de semanas.
- Pensar que la fase de pre-programación de diagramar UML y las actividades de diseño son el momento para definir de forma completa el diseño y modelar en gran detalle, y de programar de forma mecánica la traducción de éstos en código.
- Tratar de planear un proyecto en detalle desde el comienzo al final, ubicando el trabajo de cada iteración; tratar de predecir especulando todas las iteraciones, y qué pasará en cada una de ellas.
- Una organización desea planes e estimaciones creíbles para proyectos antes de que se haya ingresado en la fase de elaboración.
- Una organización piensa que adoptar RUP significa hacer la mayor cantidad de actividades posibles y crear muchos documentos, y piensan de las experiencias de RUP como un proceso formal con muchos pasos a seguir.

Ejemplo de implementación

A continuación se presenta un caso de estudio sobre la implementación de RUP en una compañía de desarrollo de software SME en Noruega con el objetivo de que esto represente mayor profesionalismo en la compañía [RUP-CASES].

La compañía se encuentra físicamente distribuida en dos oficinas y se especializa mayormente en el área de bancos y finanzas. Posee aproximadamente 50 proyectos en curso.

Los desarrolladores son mayormente especialistas en Java y J2EE. Poseen cuatro empleados certificados en RUP.

Se utilizaron dos fuentes de datos, entrevistas a los Project Manager (PM) representantes de cuatro proyectos donde cada uno debió completar una planilla con los artefactos, roles y actividades intervinientes marcando con colores si se emplearon tal como lo especifica RUP o si efectuaron modificaciones. La segunda fuentes fueron entrevistas a otro cinco empleados (desarrollador, desarrollador/PM, desarrollador/PM/Tester Manager(TM), PM/Ingenieros de requerimientos y contactos de cliente.

El análisis de datos consistió en el análisis de las planillas entregadas por los PM y el análisis de entrevistas.

Entre algunas de las conclusiones que se manifiestan podemos mencionar:

- Es necesario buen conocimiento acerca de RUP para poder implementarlo. Más capacitaciones y un foro donde los PMs puedan compartir experiencias y aprender.
- Evitar el uso de RUP en sistemas de mantenimiento.
- Los proyectos web necesitan un soporte especializado que RUP no aportó.
- Algunos elementos RUP dejaron de utilizarse debido a la falta de conocimiento acerca de cómo llevarlos a cabo.
- Si bien RUP lleva a cabo todas las fases del ciclo de vida de desarrollo de un proyecto, el cliente hace parte del trabajo con prácticas internas lo cual a futuro podría traerles inconvenientes.
- RUP es demasiado abarcativo y abrumador para pequeños proyectos.
- El uso de RUP podría haber sido mejor si se hubiese introducido de forma más cuidadosa y guiada en cada proyecto.

Luego de finalizado el caso de estudio, la empresa decidió iniciar un proceso de adaptación para proveer a sus empleados mejor soporte en el proceso.

En este caso, si el objetivo era emplear RUP como metodología de desarrollo de software se tendría que haber hecho un análisis para adaptarlo a cada necesidad de proyecto ó utilizando prácticas ágiles también posibles de introducir cuando se utiliza RUP, sobre todo para desarrollos web.

3.1.5 Jackson Development Methods

Según M. A. Jackson un método de desarrollo es un procedimiento por el cual partiendo de un problema se llega a una solución. Independientemente de que los métodos sean algorítmicos o no, deben permitir descomponer las tareas de desarrollo en un número razonable de pasos por medio de los cuales el desarrollador pueda determinar si se está

llegando a una buena solución. Éstos pasos indican de alguna forma, un orden en las decisiones que se toman durante el desarrollo de un proyecto [JACKSON-SDM].

Los métodos Jackson son particularmente adecuados para problemas donde lo que interesa es la secuencialidad ordenada en el tiempo.

Los métodos de desarrollo Jackson son JSP (Jackson Structured Programming) y JSD (Jackson System Development).

A continuación se describe cada uno de ellos.

3.1.5.1 Jackson Structured Programming (JSP)

Jackson Structured Programming o Programación Estructurada Jackson, es un método para diseño de programas en forma de composición de procesos secuenciales [JACKSON-JDM].

Fue originalmente desarrollado en los años setenta por Michael A. Jackson y documentado en su libro en 1975 llamado Principios de Diseño de Programas (Principles of Program Design). Si bien apuntó a mejorar la programación estándar de Cobol[8], su método todavía es utilizado en la codificación de lenguajes de programación, entre ellos C y Perl[9]. Originalmente fue dirigido por medio de la escritura de programas de procesamiento de archivos estilo batch, sus principios se usan en los métodos de programación orientada a objetos[10].

Con el fin de crear programas que luego sean fáciles de modificar las entradas, salidas y estructuras internas del programa construido usando JSP concuerdan; entonces pequeñas modificaciones en las entradas y salidas deberían corresponderse con cambios menores en el programa[11].

JSP define la entrada de sus programas como cuatro tipos de componentes: operaciones, secuencias, iteraciones y selecciones. Las estructuras usan expresiones regulares debido a que los lenguajes regulares proveen las tres construcciones básicas: secuencia, selección e iteración [JACKSON-JDM].

Las entradas y salidas son modeladas en Diagramas de Estructuras de Datos separados (DSD). Una vez ingresadas, las entradas y salidas son unificadas en una estructura de programa llamada Diagrama de Estructura de Programa o PSD, que luego será implementada en un lenguaje de programación, Fortran, Pascal, Cobol, etc. Esta operación puede involucrar estructuras de control de alto nivel. De acuerdo al tipo de

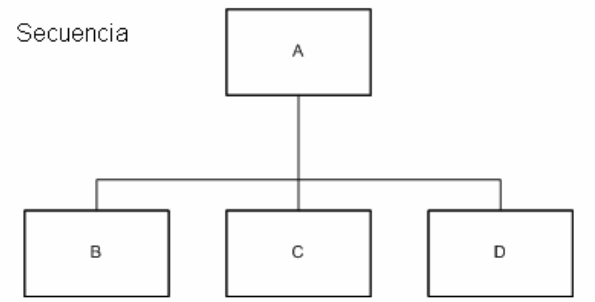
[8] http://en.wikipedia.org/wiki/COBOL
[9] http://en.wikipedia.org/wiki/Perl
[10] http://en.wikipedia.org/wiki/Jackson_Structured_Programming
[11] http://www.answers.com/Jackson%20Structured%20Programming

programa, algunos no generan la salida hasta tanto no se hayan procesado todas las entradas, otros en cambio leen en un registro, escriben y luego iteran.

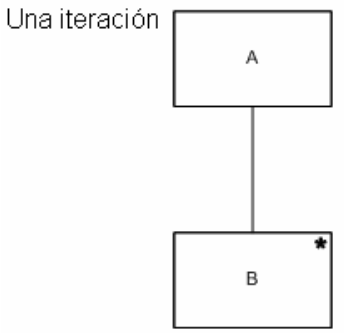
Los programas a diseñar deben procesar una o más de una secuencia (stream) de datos en forma secuencial. La secuencialidad debe ser espacial o temporal.

JSP usa diagramas para describir las estructuras de entradas, salidas y programas. Las secuencias de datos se representan con círculos y el programa con un rectángulo.

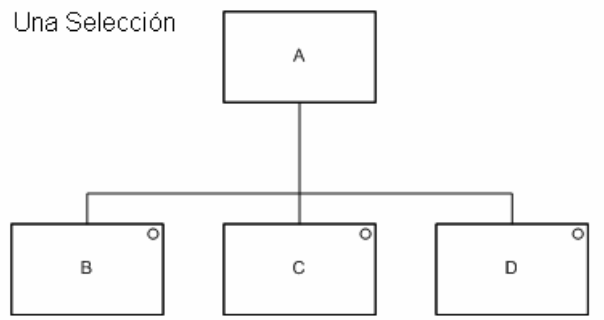
Una secuencia de operaciones se representa mediante rectángulos unidos por líneas. En el diagrama que se muestra debajo se indica que la estructura A consiste de una secuencia de operaciones (B,C y D).



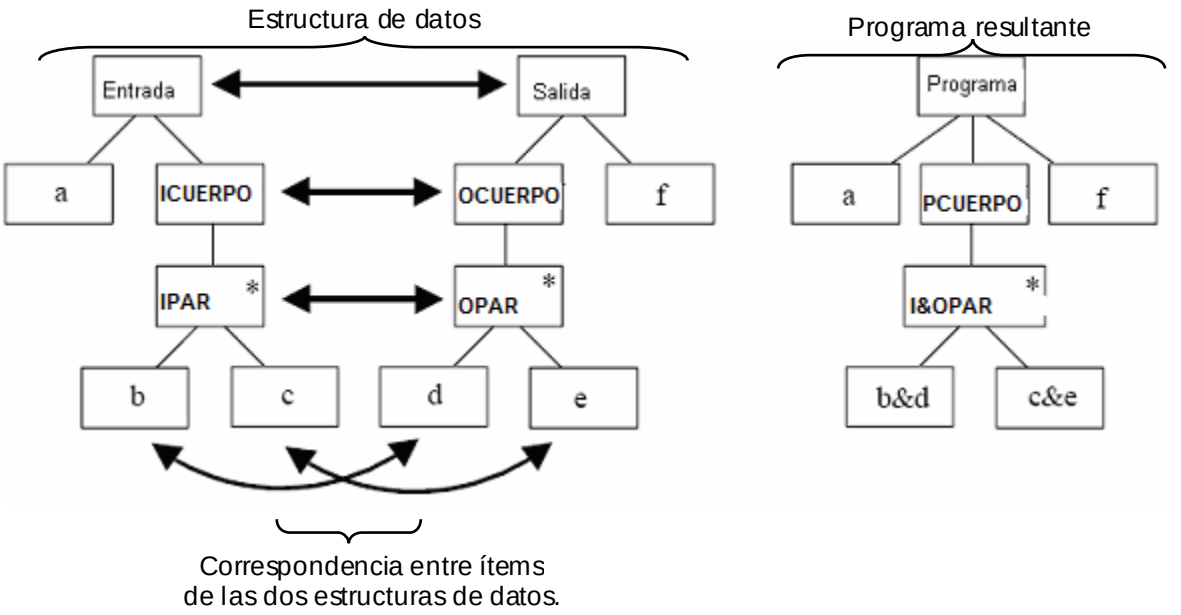
Una iteración se representa mediante la unión de dos rectángulos y para diferenciarla de las secuencias, se le agrega un asterisco en la esquina superior derecha del rectángulo. Debajo se muestra una iteración de 0 o más invocaciones a la operación B.



Por último, para representar una selección se agrega un círculo en la parte superior derecha de cada operación opcional. Debajo se muestra que la A consiste de una y solo una operación B, C o D.



A continuación se incluye un ejemplo extraído de [JACKSON-JDM] que muestra las entradas, las salidas y el programa producto de la unión de ambas estructuras. Con * se indican las iteraciones (IPAIR y OPAIR). El programa, como cualquier otro en JPS, puede pensarse como una trasversal sincronizada de dos estructuras de datos en las que sus ítems son unidos (b con d y c con e).



En Wikipedia[12], se puede encontrar otro ejemplo práctico de implementación JSP.

3.1.5.2 Jackson System Development (JSD)

Jackson System Development ó Sistema de Desarrollo Jackson, es un método para especificación y diseño de sistemas cuyos dominios de aplicación son de tipo temporal y

[12] http://en.wikipedia.org/wiki/Jackson_Structured_Programming

pueden ser descriptos usando secuencia de eventos. Debido a que las técnicas empleadas en el desarrollo de sistemas también pueden ser aplicadas en el desarrollo de los mismos, Jackson basó JSD en JSP [JACKSON-JDM].

JSD fue el primer método de diseño de sistemas "Orientado a Objetos" donde los objetos eran los procesos de comunicación secuenciales.

JSD puede comenzar desde una etapa en el proyecto donde solo hay una declaración general de requerimientos, sin embargo también existen experiencias donde comienza en etapas posteriores del ciclo de vida de un proyecto a partir de documentación preexistente. En las primeras etapas de JSD, se produce un conjunto de diseños de problemas de programa y luego, en los últimos pasos se genera el código del sistema final [JACKSON-JSD].

Principios:

Según Jackson en [JACKSON-SDM] los tres principios básicos de JSD son:

- El desarrollo debe comenzar describiendo y modelando el mundo real en vez de especificar o estructurar la función que el sistema cumple. Por ello primero se crea una simulación del mundo real donde se construye primero el modelo y luego se agregan las partes del sistema que producirán la salida.
- Un modelo adecuado dirigido por tiempo debe asimismo ser ordenado por el tiempo.
- El sistema debe ser implementado transformando la especificación en un conjunto de procesos eficientes, adaptados para correr sobre el software y hardware disponible.

Proceso:

Un proceso secuencial consiste en una instancia de ejecución de un programa.

JSD puede dividirse en etapas y subetapas.

Modelado

En esta etapa de modelado, los desarrolladores analizan el negocio tomando solo cuestiones relevantes y considerando cómo debería ser la organización. Lo relevado es detallado en una descripción bien precisa. Para lograr el grado de detalle requerido es necesario que todos los actores interesados en el sistema estén involucrados.

Se crea una colección de diagramas de entidades estructurados elaborados según la diagramación de JSP donde se identifican: entidades, acciones y el orden temporal de las mismas en la vida de las entidades y sus atributos[13].

Cada **acción**: tiene un tiempo en el que toma lugar (un proceso no puede ser una acción porque toma lugar sobre un período de tiempo), debe tomar lugar en el mundo real fuera del sistema y no se puede descomponer en subacciones.

Cada **entidad**: ejecuta o finaliza una acción en un tiempo, debe existir en el mundo real al igual que las acciones, y debe ser individual.

El resultado de esta etapa es un conjunto de tablas, definiciones y diagramas que describen en términos de **usuario**: lo que sucede en la organización y qué tiene que ser conservado sobre lo que sucede y por último, en términos de **implementación**: el contenido de la base de datos, restricciones de integridad y reglas de actualización [JACKSON-JSD].

Red

Se define la funcionalidad del sistema (lo que el sistema hace y las salidas que produce) en términos de redes de programas (comunicación secuencial de procesos). Para construir estas redes se van agregando nuevos programas y se los conecta a la red existente mediante dos formas de conexión: por streams de datos que se representan con círculos y por inspección de vectores de estado representados con rombos.

Los diagramas se complementan con información en forma de texto que describen el contenido de los streams de datos y las conexiones con vectores de estado. Los nuevos programas que se agregan a la red son definidos usando la misma notación y son diseñados usando el método JSP [JACKSON-JSD].

Implementación.

Se enfoca en cómo los procesos son orquestados y cómo los datos (vectores de estado de procesos) deben ser organizados y manejados. Para estos se utilizan técnicas de JSP.

En esta etapa se pasa de un modelo abstracto de red a un sistema físico representado en un **diagrama de implementación del sistema** que muestra al sistema como un orquestador de procesos que invoca módulos que implementan procesos. Los streams de datos son representados como llamadas a procesos invertidos, los símbolos de base de datos como colecciones de entidades vectores de estado y los buffers de archivos con símbolos especiales.

[13] www.en.wikipedia.org/wiki/Jackson_System_Development

3.2 METODOLOGÍAS ÁGILES DE DESARROLLO DE SOFTWARE

En esta sección se estudian las metodologías de desarrollo ágiles. En primer lugar se explicará el surgimiento de las mismas, sus características en general y se hará una comparación con las metodologías tradicionales estudiadas en la sección anterior. Se hará una revisión de las ventajas que aportan y las limitaciones que poseen.

3.2.1 Introducción.

Habiendo presentado un breve detalle de las metodologías tradicionales de software y avanzando también en la evolución de las mismas, se describirá las metodologías ágiles de desarrollo de software. Su surgimiento, características, comparaciones con las tradicionales, ventajas y limitaciones.

Si bien se indicarán las fortalezas y debilidades de cada una, se tendrá en cuenta que, tal como dice Hawrysh y Ruprecht, una simple metodología no puede funcionar para todo el espectro de diferentes proyectos. Un director de proyecto deberá identificar la naturaleza del proyecto y luego seleccionar la metodología de desarrollo de mejor aplicación para éste.

3.2.2 Surgimiento

El término ágil aplicado al desarrollo de software nace en febrero de 2001, tras una reunión en Utah, Estados Unidos donde participaron 17 grupos representantes de diferentes métodos ágiles de la industria del software, como ser Extreme Programming (XP), SCRUM y Crystal. De ésta forma surgió la Alianza Ágil[14], una organización sin fines de lucro y dedicada a promover los conceptos relacionados con el desarrollo ágil de software y su aplicación en las organizaciones. Si bien no se pretendía especificar tácticas detalladas de determinados proyectos, dado que los miembros eran representantes de diversas compañías, aceptaban los valores que soportaba el desarrollo de software con metodologías ágiles y lo expresaron en el Manifiesto Ágil.

El objetivo principal, entonces fue elaborar una serie de principios que permitieran a los equipos de desarrollo de software operar rápidamente adaptándose a los cambios que pudiesen surgir en cada proyecto.

[14] http://www.agilealliance.org

El origen y las buenas prácticas de los métodos ágiles son los sistemas de información tradicionales y las prácticas de implementación.

Cuando se trata de proyectos donde se exige reducir drásticamente los tiempos de desarrollo manteniendo la calidad esperada, el enfoque "tradicional" no es el más indicado dado que no permite una rápida adaptación a un ambiente marcado con un alto nivel de cambios, genera gran cantidad de datos y al mismo tiempo poca información de control real. Además debido a esto generalmente se multiplican los presupuestos y se tarda mucho más tiempo del estimado inicialmente.

Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchas consultoras se preocupan por vender horas y no son castigadas por incumplimiento, mientras que otras compañías de software deben entregar productos terminados y no cobran lo merecido o son penalizadas.

En este escenario, las metodologías ágiles nacen como respuesta al fracaso en llevar a proyectos de desarrollo de software a buen término y en forma.

Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto [Metodologías Ágiles: José H. Canós, Patricio Letelier y Mª Carmen Penadés].

Actualmente existen 4 conferencias internacionales de alto nivel y específicas sobre el tema [Conferencias].

Una de las fortalezas más importantes de éste tipo de metodologías está dada por su capacidad de ser ajustable a una amplia gama de proyectos de desarrollo de software donde los equipos de desarrollo son pequeños, con tiempos de entrega reducidos y a veces con el requerimiento de aplicación de nuevas herramientas y cambios en los recursos.

3.2.3 El Manifiesto Ágil.

El Manifiesto Ágil, es un documento que expresa la filosofía "ágil" y su distinción de los métodos tradicionales. Según el Manifiesto Ágil se valora lo siguiente:

- Los individuos y sus interacciones son más importantes que procesos y herramientas. Lo importante es satisfacer al cliente a través de la entrega temprana y continua de software con valor.
- El software que funcione es más importante que la documentación exhaustiva.
- Colaboración con el cliente en lugar de negociación de contratos.
- Repuesta ante el cambio en vez de seguir un plan cerrado.

Los principios definidos en el Manifiesto Ágil son[15]:

- Nuestra mayor prioridad es satisfacer al consumidor por medio de entregas continuas de software con valor.
- Son bienvenidos los requerimientos que implican cambios, aún en una etapa tardía de desarrollo.
- Entrega frecuente de software funcionando desde un par de semanas a un par de meses, preferentemente en una escala a corto tiempo.
- Los responsables de negocio y los desarrolladores deben trabajar de forma conjunta diariamente a lo largo del proyecto.
- Construcción de proyectos con individuos motivados. Darles el ambiente y soporte que necesiten y confiar en ellos para tener el trabajo hecho.
- El método más efectivo y eficiente de reunir información entre los equipos de desarrollo es con conversaciones cara a cara.
- El software funcionando es la primera medida de progreso.
- Los procesos ágiles promueven el desarrollo sustentable. Los patrocinadores, desarrolladores, y usuarios deben ser capaces de mantener de forma indefinida una constante paz.
- La continua atención a la excelencia técnica y al buen diseño realza la agilidad.
- La simplicidad, como arte de maximizar la cantidad de trabajo no hecho, es esencial.
- Las mejores arquitecturas, requerimientos y diseño surgen de equipos auto-organizados.
- En intervalos regulares, los equipos indican cómo lograr una mayor efectividad, luego mejoran y ajustan su comportamiento de forma acorde.

3.2.4 Características.

Según [AGILE-METHODS], lo que hace que un método de desarrollo sea ágil es la simplicidad y la velocidad. En el trabajo de desarrollo, el grupo de desarrollo solo se concentra, en primera instancia, en las funcionalidades básicas del sistema, entregando versiones de forma rápida, recibiendo feedback y actuando luego en base a esto. Por lo tanto una metodología de desarrollo ágil se puede describir como un desarrollo de software *incremental* (pequeñas versiones con ciclos rápidos), *cooperativa* (clientes y desarrolladores

[15] http://www.agilealliance.org/principles.html

trabajando constantemente juntos), *sencillo* (método fácil de aprender, modificar y documentar) y *adaptativa* (capaz de impactar cambios de último momento).

Hightsmith y Cockburn en el año 2002 indicaron que el cambio en los negocios del software, afecta al proceso de desarrollo de software. A su vez, satisfacer a los clientes en el momento del deploy adquirió mayor precedencia que hacerlo al momento de la iniciación del proyecto. Esto requerirá la llamada a procedimientos que no se enfoquen en cómo detener los cambios sino encontrar la mejor forma para manejar los cambios inevitables en el ciclo de vida del proyecto. Por ello los métodos ágiles están diseñados para [AGILE-METHODS]:

- Producir una versión prototipo de aplicación en las primeras semanas y así obtener feedback del cliente.
- Inventar soluciones simples de modo que luego sea fácil efectuar cambios sobre ellas.
- Mejorar la calidad del diseño continuamente para lograr que la siguiente versión sea menos costosa.
- Hacer pruebas a menudo con el fin de lograr la detección de fallas de forma temprana y a menor costo.

Ambler, en el año 2002, definió un conjunto de aproximaciones que surgen de los procesos de desarrollo ágiles:

- Lo que importa son las personas.
- Lograr la menor cantidad de documentación posible.
- La comunicación es una tarea crítica.
- Las herramientas de modelado no son tan útiles como parecen.
- No es requerido un gran diseño up-front (ver Glosario).

Por su parte, Jim Highsmith describe las tres principales características de una metodología ágil [AGILE-QES].

1) *Manejo adaptativo*. Crear un ambiente con variedad de requisitos para enfrentar el desafío de proyectos extremos y particularmente los de grandes cambios.
2) *Valores y principios colaborativos*. Muchas metodologías son diseñadas para estandarizar las personas a los proyectos, mientras que las ágiles lo son para capitalizar las habilidades individuales y en equipo, y de esta forma, adaptar los procesos a las personas.
3) *Apenas lo suficiente*. Intenta resolver la pregunta de cuánta estructura es suficiente. La idea es lograr un balance entre flexibilidad y estructura de forma tal que se reduzcan los costos y sobre todo, hacer hincapié en que la creatividad y la innovación ocurren en ambientes no estructurados.

3.2.5 Comparación con las metodologías "tradicionales".

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso controlado en primera instancia por objetivos de negocio y por actores principales, y menos controlado artificialmente por conceptos externos a los actores principales.	Proceso mucho más controlado, con numerosas políticas/normas que pueden llegar impedir determinadas acciones.
No existe contrato orientado al éxito de proyecto. Es un contrato flexible. La entrega de la aplicación funcionando podría tomarse como contrato.	Existe un contrato prefijado y menos flexible. Los contratos se basan en tamplates generados por las consultoras. Y en el caso en que el cliente quiera efectuar un cambio en el mismo, la consultora deberá aprobarlo.
El cliente es parte del equipo de desarrollo.	Normalmente el cliente interactúa con el equipo de desarrollo mediante reuniones. Esto no siempre se da, muchas veces sucede que los que asisten a las juntas son solo los responsables del proyecto y finalmente el equipo desconoce al cliente con poder de decisión.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
La arquitectura del software esta supeditada al proyecto, y no de forma inversa.	La arquitectura del software es esencial y se expresa mediante modelos. Muchas veces la arquitectura dicta el proyecto e incluso al negocio.
Todas las personas tienen un perfil multi-rol, esto quiere decir que todos son dueños de todo y a su vez responsables de todo.	Personas mono-rol.
La validación de los requerimientos se efectúa al final de cada iteración [AGILE-REQ].	La validación de los requerimientos se efectúa antes de la implementación [AGILE-REQ].

3.2.6 Ventajas.

Si bien los procesos Ágiles, como XP (Extreme Programming) se enfocan en la construcción de productos de software que resuelvan un problema específico, también permiten el **desarrollo de componentes genéricos y reusables**. Un ejemplo es el caso del desarrollo de software libre.

Por otra parte, el proceso Ágil de desarrollo soporta el manejo de proyectos donde la coordinación, el control y los mecanismos de comunicación son aplicados en equipos chicos o medianos. Si bien algunos autores en sus trabajos, por ejemplo [AGILE-LIM] no concuerdan en que el uso de **metodologías ágiles también puede aplicarse para grandes proyectos**, otros indican lo contrario, diciendo que esto es posible pero mediante la conformación de equipos de menor tamaño donde se puedan mantener las comunicaciones necesarias (como son las cara a cara y reuniones de revisión). Esto nos demuestra que éste tipo de metodologías es adaptable a proyectos de diferentes características. Como un ejemplo adicional, se puede citar a GNU/Linux, donde los equipos están distribuidos por todo el mundo, el acoplamiento es suave y difuso, y conforman un equipo de desarrollo mucho mas productivo y eficiente que otras empresas más poderosas.

Según Michael Vax[16] y tal como se fue mencionando en las secciones anteriores donde se explican las características de las metodologías de desarrollo ágiles, se pueden mencionar otras ventajas:

- El desarrollo de software con metodologías ágiles permite la producción de software en pequeñas iteraciones. Minimiza la planificación en etapas iniciales. Al tener una planificación en cada etapa de desarrollo se tiene una metodología más fluida que garantiza mejores resultados.
- Al predecir solo con unas pocas semanas de anticipación, se minimiza la pérdida de tiempo que insumiría la documentación. Se tienen en cuenta las variables de todos los días que pueden afectar cualquier proyecto, por ejemplo cambios de personal, desarrollos tecnológicos y cambios en regulaciones como son las del sector financiero.
- Debido a las constantes integraciones y pruebas en cada etapa, el producto final puede versionarse lo antes posible, no bien está listo, y los cambios pueden realizarse de forma tal que el desarrollo lleve a una implementación más rápida y exitosa.
- Las pruebas ejecutadas en cada iteración aseguran que las fallas puedan ser corregidas regularmente. Por otra parte, al desarrollar las pruebas de esta forma se

[16] http://www.computerworlduk.com/whitepapers/ Michael Vax. Agile methodology: Why aren't you using it?

establece una relación colaborativa y más cercana con el usuario quién ayudará a detectar las fallas y conocerá las correcciones que incluirá la siguiente iteración.

- En relación con el trabajo en iteraciones de corta duración y el versionado rápido, se pude **aprender** más, debido a que constantemente se ven los resultados que ayudan a saber qué es lo que se quiere y lo que se necesita. Tanto usuarios como clientes y demás, pueden participar en la determinación de lo que se está construyendo. No solo permite el aprendizaje acerca del producto sino que también acerca del proceso de construcción, es decir, cada vez que se pasa de iteración en iteración se aprende cómo hacer el trabajo de forma más efectiva y eficiente y al final de cada iteración se puede cambiar el estilo de trabajo si se lo considera conveniente[17].
- Provee visibilidad y frecuencia de entrega[18]. Visibilidad dado que permite a cualquier integrante ver las demos, examinar los diagramas o tableros de control de tareas, preguntar a los miembros del equipo inquietudes, etc. Las entregas con frecuencia a través de las iteraciones cortas permiten a los usuarios la oportunidad de ver el progreso y efectuar cambios.
- Rápido ROI (Retorno de Inversión)[19]. La estructura básica de los métodos ágiles consiste en ciclos cortos de entrega de software con resultados de valor[20] en cada iteración proveyendo esto las bases para un rápido retorno de inversión. La organización puede tomar estos resultados para medir sus beneficios idealmente luego de la primera iteración.

A continuación se incluye un gráfico que compara de la metodología Waterfall y de las Ágiles en tiempos y costos.

17 http://del.icio.us/post?url=http://www.agileadvice.com/archives/2007/09/agile_benefits.html&title=Agile%20Benefits:%20Rapid%20Learning

18 http://del.icio.us/post?url=http://www.agileadvice.com/archives/2007/09/agile_benefits_3.html&title=Agile%20Benefits:%20Satisfied%20Stakeholders

19 http://del.icio.us/post?url=http://www.agileadvice.com/archives/2007/09/agile_benefits_2.html&title=Agile%20Benefits:%20Early%20Return%20on%20Investment

20 http://www.xprogramming.com/xpmag/jatRtsMetric.htm

Con la línea roja se muestra cómo con Waterfall se aporta todo el valor al final, mientras que con metodologías ágiles se incrementan al máximo lo antes posible las entregas de valor y luego se reducen gradualmente. Esto se debe a que al comienzo de un proyecto existe incertidumbre en gran medida, se deben resolver cuestiones prioritarias antes de alcanzar el trabajo de valor , pero luego de que esto es completado, el equipo comienza a trabajar menos y con funcionalidades de menor prioridad. Las entregas de valor anticipadas permiten que el valor de esa inversión sea incrementado al entregarse más funcionalidad en menor tiempo. El rédito se puede alcanzar de forma más rápida y por consiguiente, se puede destinar más fondos a otras inversiones.

3.2.7 Limitaciones.

En el estudio llamado “Limitaciones de los Procesos de Software Ágiles” en inglés “Limitations of Agile Software Processes” [AGILE-LIM], se incluye una serie de situaciones para las cuales, éste tipo de metodología de desarrollo no puede ser aplicable. A su vez indica que es posible que algunos procesos ágiles encajen mejor para estas consideraciones mientras que otras quizás requieran de agregar principios asociados con otras de desarrollo predictivas. A continuación se detallan éstas situaciones extraídas de [AGILE-LIM].

1) **Soporte limitado para ambientes de desarrollo distribuido.**

Los procesos de desarrollo ágiles recomiendan reuniones cara a cara de forma periódica. Cuando el equipo de trabajo y los clientes están distribuidos físicamente esto se vuelve dificultoso. En estos proyectos, estas reuniones son reemplazadas por videoconferencias. Además de que ésta es una tecnología cara y no siempre permite lograr el resultado esperado, surge la necesidad de generar entre cada encuentro, documentación de requerimientos y diseño que luego debe ser mantenida conforme al paso del tiempo y posteriores reuniones, con el objetivo de asegurar que todos los integrantes del equipo tengan la misma visión del producto a construir.

2) **Soporte limitado en subcontratos.**

Cuando se trata proyectos donde se solicita a una empresa extranjera un determinado desarrollo, es necesario que esta empresa interesada presente una licitación que incluya un plan con procesos, hitos y entregas, con el suficiente detalle a fin de determinar un costo estimado. Si bien los procesos pueden ser iterativos y con una aproximación incremental, el contratista deberá hacer los procesos predictivos, especificando el número de iteraciones y

entregas por cada iteración hasta completar el producto. Es posible que este contrato sea escrito de forma que permita algún grado de flexibilidad en cómo se desarrolla el producto dentro de las restricciones de tiempo y costos.

Un contrato soportado por el desarrollo Ágil debe contener dos partes: en la primera (llamada fija) se define la forma en que el contratista incorporará los cambios en el producto, las actividades que deberán llevarse a cabo y los requerimientos a entregar. En la segunda parte (llamada variable), se definirán los requerimientos y entregas que pueden variar dentro de los límites definidos en la parte anterior. Pueden desarrollar las restricciones definidas en ella y al momento en que se firma el contrato, una descripción de las entregas prioritarias y requerimientos que deberán incluirse.

3) Soporte limitado en el desarrollo seguro de software crítico (ver Glosario).

Los mecanismos de control de calidad soportados por los procesos Ágiles, todavía no han probado ser adecuados y suficientes para asegurar a los usuarios que el producto sea seguro.

Sin embargo existen técnicas Ágiles sugeridas que brindan beneficios al desarrollo de éste tipo de software como ser: escribir los casos de prueba antes de codificar, poner que soporte el desarrollo exploratorio de software crítico para los cuales los requerimientos no están bien definidos, y por último la programación de a pares como suplemento para las revisiones formales.

4) Soporte limitado en el desarrollo de sistemas grandes y complejos.

En los sistemas complejos o de gran tamaño existen aspectos de arquitectura críticos que pueden dificultar un cambio, debido al rol crítico que desempeñan en los servicios del sistema. Por consiguiente, en estos casos puede aumentar en gran escala el costo y luego implicar grandes esfuerzos por anticipar dichos cambios. Bajo este escenario, optar por reescribir código podría traer problemas, debido a la complejidad y el tamaño del sistema, pudiendo obtener código estricto y costoso que provoque fallas. Ante esta situación, se resalta el papel importante que tienen las herramientas que generan código a partir de los modelos.

El autor indica que en algunos casos pueden existir sistemas donde la funcionalidad está altamente acoplada e integrada de forma tal que impide el desarrollo de forma incremental. En esos casos sugiere utilizar proceso iterativo donde el código producido en cada iteración, incluya piezas en diferentes estados de in completitud.

No obstante esto, vale la pena indicar que la única forma de lograr el desarrollo de sistemas grandes y complejos es mediante un proceso incremental e iterativo, mediante la generación de prototipos simples y pequeños que conformarán pequeñas partes del sistema

que serán luego ensambladas de forma que puedan completar la total funcionalidad del sistema.

Según Michael Vax, podría provocar confusión y consternación si se aplica una metodología ágil en organizaciones con estructuras rígidas donde los proyectos son planificados estrictamente desde comienzo a fin. Sugiere también que podría traer complicaciones serias si el cliente no está altamente involucrado en el desarrollo ó por el contrario si el cliente está altamente involucrado pero sugiere demasiados cambios que provocarían un crecimiento mucho mayor que el alcance original.

3.2.8 Metodologías Ágiles:

Si bien los métodos ágiles comparten características comunes, como ser principios y valores, difieren en las prácticas que sugieren. A continuación se describirán brevemente los siguientes métodos ágiles: Métodos Cristal, Scrum, Dynamic Systems Development Methodology, Lean Development, Feature Driven Development y Extreme Programming.

3.2.8.1 Extreme Programming

Extreme Programming (Programación Extrema ó XP) es una disciplina de desarrollo de software con valores de simplicidad en las soluciones implementadas, comunicación entre las personas involucradas fortaleciendo las relaciones interpersonales, realimentación continua entre cliente y equipo de desarrollo, y coraje para poder enfrentar de forma adecuada los cambios.

Fue desarrollada por Kent Beck en 1999 en [XP-BECK] sin cubrir detalles técnicos ni de implementación de las prácticas, pero luego éstas fueron desarrolladas.

Se enfoca en los roles de cliente, manager y programador y acuerda derechos clave y responsabilidades para esos roles [XP-INSTALL]. A continuación se describen éstos roles.

El rol de **cliente** escribe las historias de usuarios y las pruebas funcionales para validar su implementación. Asigna prioridades a las historias de usuario y decide cuáles se implementarán en cada iteración de forma tal que aporte mayor valor al negocio.

El rol de **programador** analiza, diseña, ejecuta pruebas, programa e integra el sistema. Estima la dificultad de todas las historias y rastrea los pasos en los cuales pueda entregar historias a los clientes. Debe construir el sistema de a pequeñas entregas, de

forma tal que los beneficios del cliente se maximicen y se obtenga el mejor feedback sobre lo que se está haciendo [XP-INSTALL].

El rol del **administrador** del proyecto une a los clientes y desarrolladores y los ayuda a ser un equipo operativo. No hacen el proceso sino que hacen que el proceso sea llevadero. Observa las cosas que retrasan el proyecto y las resuelve. Quita del camino todo lo que no contribuye con el objetivo de entregar buen software y en tiempo. Hace que se cumpla la planificación, el diseño y las pruebas, la codificación, las entregas coordinándolas y reportando los resultados [XP-INSTALL].

XP es especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico [XP-BECK].

Tiene un enfoque adaptativo con lo cual, la planificación del proyecto progresará a medida que los cambios vayan surgiendo. Se concentra en la fase de implementación con énfasis en la entrega de software en funcionamiento.

Se basa en la idea de que existen cuatro variables, las cuales guían el desarrollo de los sistemas: costo, tiempo, calidad y alcance. XP permite a las fuerzas externas (gerencia y clientes) manejar hasta tres de estas variables, quedando el control de la restante en manos del equipo de desarrollo. XP hace visibles estas cuatro variables [XP-BRCONS].

XP se propone disminuir la curva exponencial del costo del cambio a lo largo del proyecto a fin de lograr que funcione el diseño evolutivo [AGILE-ISSI].

Según [XP-REFACT] en XP no hay fases. El diseño se desarrolla de acuerdo a los requerimientos. El código es desarrollado de acuerdo al diseño. Así es que los requerimientos conducen la codificación; el código es el diseño. Todo eso ocurre al mismo tiempo.

Debido a que la base del código es desarrollada y la planificación, el diseño, la construcción y las pruebas suceden simultáneamente, en lugar de ser el sistema planeado, diseñado por completo y luego construido en fases separadas, a este proceso se le denomina desarrollo evolutivo. Esto contrasta con otros procesos como ser RUP, los cuales tienden a ser iterativos e incrementales. Con XP el proyecto es dividido en iteraciones de 1 a 3 semanas, con las entregas manejadas por el cliente al final de cada iteración (cada cliente determinará si una entrega será liberada para el usuario final.

A su vez, posee un estilo en cascada en micro-fases. El ciclo de vida “Diseño de requerimientos → Código → Integración → Pruebas → Deploy” toma lugar en una simple iteración (1 a 3 semanas). Las “fases interiores” son repetidas varias veces en el día y los

límites entre diseño, codificación y pruebas son llevadas a tal punto que son casi indistinguibles. Los requerimientos son referidos como Exploración y el propósito es la identificación, priorización y estimación de los mismos, que suele darse en el primero o segundo día de cada iteración [XP-REFACT].

Para [AGILEM-XP], el ciclo de vida ideal de XP consiste en seis fases: Exploración, Planificación de la entrega, Iteraciones, Producción, Mantenimiento y Muerte del proyecto.

Exploración:

Los clientes plantean las historias de interés para la primera entrega y el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. Esta fase toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

Planificación de la entrega:

Una vez establecidas las prioridades de entrega de cada historia, se realizan las estimaciones de tiempo necesarias para cada una. Las estimaciones de esfuerzo asociados a la implementación realizadas por los programadores se efectúan teniendo en cuenta el punto, que equivale a una semana ideal de programación. Generalmente las historias valen de 1 a 3 puntos. A su vez, el equipo mantiene un registro de la velocidad de desarrollo. Esto sirve para la planificación por tiempo donde se multiplica el número de iteraciones por la velocidad del proyecto, determinando cuántos puntos se pueden completar. Al planificar según el alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación [AGILEM-XP].

Esta fase dura unos pocos días.

Iteraciones:

El plan de entrega se compone de no más de tres semanas. En la primera se intenta establecer una arquitectura para el sistema, mediante la elección de historias que fuercen la creación de ésta arquitectura, aunque esto no sea siempre posible.

Producción:

Aquí se requieren pruebas y revisiones antes del despliegue en el entorno del cliente. Se evalúa la incorporación de cambios que surjan durante esta fase.

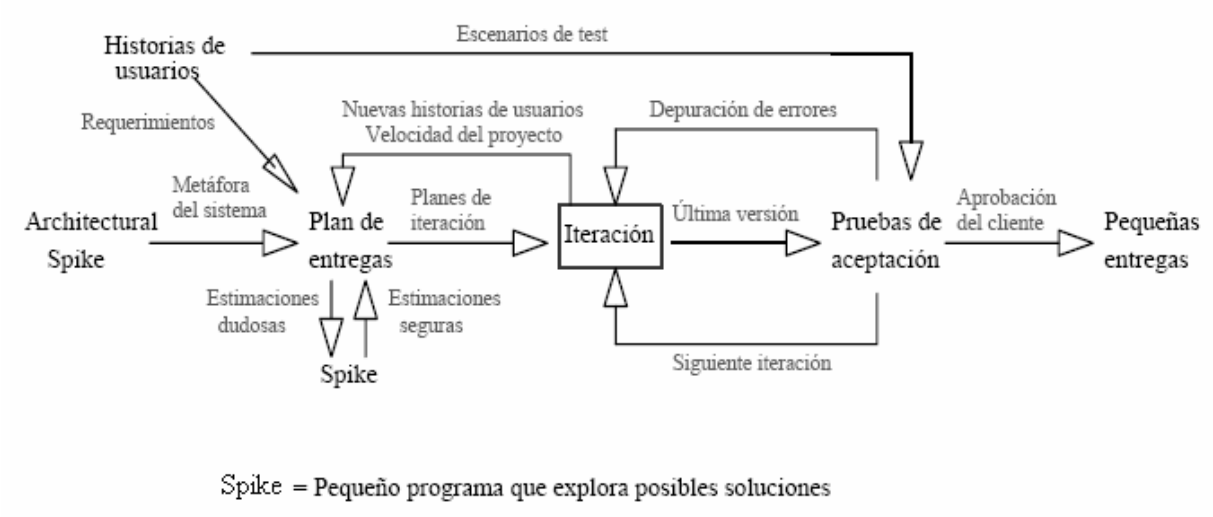
Mantenimiento:

Se debe mantener en funcionamiento el sistema que está en producción mientras se desarrollan nuevas iteraciones mediante tareas de soporte, por lo tanto se puede necesitar que más personas se involucren en estas tareas que en desarrollo.

Muerte del proyecto:

Ya no se tienen más historias para incluir, solo es necesario cumplir con las tareas de mantenimiento y rendimiento. Se finaliza la documentación del sistema y no se hacen más cambios en la arquitectura.

A continuación se incluye un diagrama que muestra las fases involucradas en el ciclo de vida del desarrollo de softw are con XP antes descriptas.



Proyecto Extreme Programming [XP-INTRO]

Ciclo de vida:

Consiste en pasos sencillos durante los cuales tanto programadores como clientes aprenden que:

1) El cliente define el valor de negocio a implementar.
2) El programador estima el esfuerzo necesario para su implementación.
3) El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4) El programador construye ese valor de negocio.
5) Vuelve al paso 1.

Prácticas.

XP propone 12 prácticas que permitirán lograr desarrollos de alta calidad en los tiempos estimados y con costos razonables. Todas estas prácticas, que no son del todo novedosas dado que en Ingeniería de Software ya fueron mencionadas, son una unidad donde cada una soporta a las demás. Por eso es recomendable seguirlas a todas [XP-BRCONS]. Siguiendo [XP-BRCONS] y [AGILE-ISSI] se mencionan y describen brevemente cada una de ellas.

1) Jugar el juego de la planificación. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración. Se determina el alcance de la siguiente versión teniendo en cuenta las prioridades del negocio y las estimaciones técnicas. Se adaptarán los planes en caso de que la realidad sobrepase lo planificado.

2) Hacer pequeñas versiones entregables. Producir rápidamente versiones del sistema que sean operativas. Cada entrega deberá ser desde una por semana a una por mes.

3) Hacer historias y usar metáforas. El sistema es definido mediante una o varias metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (vocabulario para hablar sobre el dominio del problema , ayudando a la nomenclatura de clases y métodos del sistema).

4) Diseñar simple. Se debe diseñar la solución de la forma más simple posible y que aún así cubra la funcionalidad requerida por el cliente en esta iteración, e implementarse en un momento determinado del proyecto.

5) Probar. Los desarrolladores escriben continuamente Pruebas Unitarias que se corren para que el desarrollo continúe. Cuando se detecta un error, su reparación pasa a ser la máxima prioridad para el programador y/o el equipo. Los clientes (ayudados por Desarrolladores) escriben Pruebas Funcionales para probar qué funcionalidades están terminadas y se están cumpliendo de acuerdo a sus expectativas.

6) Reconstrucción (refactoring en inglés). Consiste en la reestructuración de código con el fin de remover código duplicado, mejorar la legibilidad, simplificar el código y hacerlo más flexible para enfrentar favorablemente cambios futuros.

7) Programación por pares. La producción de código se debe realizar mediante parejas de programadores (es decir esta es la unidad básica de recurso de desarrollo) frente a una misma estación de trabajo, lo que aporta los siguientes beneficios: disminuye la tasa de error, mejora el diseño, genera mayor satisfacción de los programadores, etc).

8) Propiedad colectiva del código: Cualquier integrante del equipo puede cambiar el código en cualquier parte del sistema en cualquier momento. La responsabilidad del ownership o propiedad, es compartida por todos los integrantes del equipo.

9) Integrar continuamente. Cada vez que se termina una tarea, las piezas de código se integran, aún varias veces por día, hasta lograr que el sistema íntegro funcione correctamente, de acuerdo a los requerimientos de cada iteración.

10) Semanas de 40 horas: Se debe trabajar un máximo de 40 horas por semana. No trabajar horas extra dos semanas seguidas. En caso de ocurrir esto, es porque existen problemas que corregir. Por otra parte esto desmotiva al equipo.

11) Cliente on-site: se debe incluir al menos un cliente real, vivo, como parte del equipo. El mismo debe estar disponible en tiempo completo e interactuar con el equipo dado que la comunicación verbal y personal es más efectiva que la escrita.

12) Usar estándares de codificación: se deberán seguir estándares de codificación para facilitar la comunicación entre programadores y mantener el código legible.

Ventajas.

La buena aplicación de las prácticas recomendadas en XP se traduce en beneficios en el desarrollo de un proyecto. Podemos decir entonces que XP favorece la entrega rápida de productos dado que pone el foco en ello, minimiza la cantidad de documentación necesaria para comenzar el desarrollo. Fomenta por ejemplo: el trabajo en equipo, la comunicación entre personas involucradas al proyecto, el empleo de patrones de software y las pruebas unitarias.

En [CASES-AGILESD] se mencionan puntos a favor en XP:

- La propiedad colectiva reduce el riesgo de pérdida de conocimiento o el cuello de botella al momento de efectuar modificaciones entre varios desarrolladores, cuando uno debe esperar a que el dueño de otra clase efectúe las modificaciones antes de que éste pueda iniciar su propio trabajo. Sin embargo puede traer inconvenientes si no es bien administrada. Las prácticas de XP (programación de a pares, codificación de estándares, pruebas automatizadas e integración continua) evitan que esto suceda y aseguran el orden ante este tipo de escenarios.
- La comunicación es un factor clave en el éxito de un proyecto. Implementando soluciones de la forma más simple se favorece la comunicación debido a que las cosas simples son las fáciles de comunicar.
- El feedback se incrementa entre lapsos de tiempo cortos favoreciendo la toma de decisiones, la administración y la dirección del proyecto a la dirección correcta.

- Incentiva los principios agilidad y adhesión a los cambios ayudando a alentar al equipo dándoles feedback sobre las decisiones, comunicando estrategias y usando soluciones simples.

Desventajas.

En [XP-AGILEMETH] se definen situaciones en las que XP puede fallar. Éstas y otras se indican a continuación.

1) Contar con la gente equivocada. Se debe contar con personas capacitadas técnicamente y con buen trato social, de forma que mantengan buena comunicación verbal y capaces de compartir con el resto de los integrantes de los equipos de trabajo. El equipo de desarrollo debe ser disciplinado.
2) Tener al cliente equivocado. El cliente es uno de los responsables máximos del éxito ó fracaso del proyecto, no necesariamente tiene que ser personal altamente capacitado en negocios o directivo, sin embargo con la responsabilidad del proyecto debe venir la jerarquía de mando, de lo contrario no podrá resolver situaciones que tienen que ver con el ambiente exterior al proyecto. También podría funcionar alguien con menor nivel pero que pase gran parte del tiempo trabajando y coordinando con otras personas con más experiencia en el negocio. Se requiere un alto nivel de participación del cliente en el proceso de desarrollo del producto. En [CASES-AGILESD] se plantea la existencia de muchas controversias respecto a este tema. Básicamente consisten en el planteo de la imposibilidad de contar con el cliente in situ durante la semana. Si bien se indica mediante una encuesta que a veces éste es uno de los principios más difíciles de cumplir, se pueden emplear técnicas de substitución, por ejemplo: disponer del cliente part-time trabajando junto al equipo de desarrollo.
3) Escasa comunicación. Esto no solo se refiere a personas con poca capacidad de comunicación, sino también ambientes donde se dificulta la interacción entre los participantes por situaciones como ser ambientes donde el personal está dividido en diversos edificios con varios pisos, o distribuidos en ciudades diferentes.
4) Mala aplicación de prácticas XP. La selección de prácticas XP a aplicar en un proyecto deben ser seleccionadas con juicio, dado que aplicar solo algunas de ellas no indica que se esté usando XP.
5) La Programación de a pares trajo mucha controversia. Se cree que este principio es difícil de concretar debido a dos razones. La primera radica en que es una pérdida de tiempo asignar a dos programadores a trabajar juntos sabiendo que uno solo es

capaz de completar la tarea. La segunda se basa en que los programadores son personas con dificultades para sociabilizarse. Sin embargo existen estudios que indican que el trabajo de a pares produce mejores soluciones, se reduce la cantidad de fallas y el trabajo se finaliza con mayor rapidez [CASES-AGILESD].

Ejemplo de implementación.

A continuación se da un breve detalle sobre un caso de estudio realizado en colaboración con la Universidad del Estado de Carolina del Norte y áreas de desarrollo de Sabre Airlines Solutions. Estos detalles han sido extraídos de [XP-CASEST2].

El equipo de desarrollo de Sabre-A fue seleccionado debido a su comportamiento ágil. La selección fue influenciada por la disponibilidad de datos, el tamaño del equipo, la cooperatividad del equipo con los investigadores y a que el equipo de investigación tenía un tiempo de trabajo limitado.

Se plantearon básicamente cinco hipótesis nulas indicando que las prácticas de XP no producían cambios en:

- H10: la calidad pre-release (como medida de defectos encontrados antes de la versión del producto).
- H20: la calidad post-release (como medida por defectos encontrados por el cliente luego del release).
- H30: productividad de los programadores (como medida por historias de usuario y líneas de código por persona por mes).
- H40: satisfacción del cliente (medida vía entrevistas y feedback del cliente).
- H50: estado de ánimo del equipo (evaluada mediante una encuesta).

Se utilizó un framework de evaluación XP (XPEF), que es un benchmark basado en ontologías para expresar información de casos de estudio. Lo que básicamente hace es guardar información del contexto del caso de estudio, la extensión sobre la cual la organización adoptó prácticas XP y el resultado de ellas [XPEVAL-FR].

El estudio consistió en comparar dos versiones, la número 3 donde se utilizó una metodología tradicional basada en waterfall durante 18 meses y la número 9 con XP.

Algunas métricas no se pudieron incluir en XPEF debido a que los investigadores no estuvieron presentes en la versión tercera y no sabían que cualquier investigación sería efectuada sobre sus productos o en su documentación.

A seis de cada diez miembros se les efectuaron entrevistas semi estructuradas durante el release nueve.

Los factores de contexto XP (XPFC) utilizaron seis categorías de factores introducidas por Jones [FC-JONES]: Clasificación de software (sistemas para control de dispositivos físicos, comercial, sistemas de información para información del negocio, tercerización (outsourcing en inglés), militar, usuario final para uso personal), sociológicas (tamaño del equipo, nivel de experiencia, dominio de experiencia, experiencia en lenguajes, experiencia de los PM, etc.), geográficas (ubicación del equipo, cantidad de clientes, ubicación del cliente, proveedores), específicas del proyecto, tecnológicas, ergonómicas y factores de desarrollo (aportada por Boehm y Turner [FC-BOHTU]).

Resultados

Se resumen los resultados obtenidos producto de la comparación de las dos versiones donde se midió cuantitativa y cualitativamente. Se efectuaron revisiones con los miembros de los equipos para revisar lo encontrado y luego recibir un feedback final acerca de su experiencia con XP. La información sobre defectos fue relevada del sistema de seguimiento de defectos de los equipos participantes. Se indican algunos beneficios y limitaciones indicadas por los integrantes de los equipos.

Beneficios

- El mayor beneficio aportado por XP fue el incrementó de la comunicación.
- Las reuniones permitieron a los desarrolladores comprender mejor el trabajo a realizar.
- Debido al incremento en la comunicación, los problemas se pudieron resolver con mayor rapidez. También ocurrió que algunos desarrolladores escuchando conversaciones de problemas de otros desarrolladores pudieron hacer aportes para su solución.
- Rápido feedback aportado por las pruebas unitarias y la programación de a pares durante la codificación y la fase de diseño.

Limitaciones

- Consideraron que si bien la programación de a pares es de valor para la resolución de problemas y la implementación de funcionalidades difíciles, a menudo el uso del tiempo les resultó ineficiente para tareas de poca importancia.
- El nivel de ruido provocado por la programación de a pares les significó distracción.

Análisis de las hipótesis

En todos los casos se rechaza la hipótesis nula y se acepta la alternativa tal como se explica debajo.

Calidad internamente (pre-release) visible. La densidad de defectos mejoró un 65%. Conclusión: cuando se opera con equipos dentro de un contexto específico, el uso de un subconjunto especificado de prácticas XP conduce a una mejora en la calidad del pre-release.

Calidad externamente visible. El número de defectos encontrados en el sistema productivo del cliente mejoró un 35%. En la versión nueve no se detectaron errores de categoría severidad 1 donde los defectos hacen que el sistema no pueda ser usado por el cliente. En la versión tres, existió un 12% de severidad 1 y un 82% de severidad 2. El nivel de severidad 2 indica que las fallan provocan el mal funcionamiento del sistema pero no impiden su uso.

La productividad del equipo se incrementó un 50% en comparación con el release tres. La reducción en la complejidad de las funcionalidades afectó esta métrica. La familiaridad del equipo con el dominio de la aplicación en el nuevo release también afectó los resultados. Las historias de usuario por persona no se pudieron medir dado que en la primera versión no se utilizaron.

Satisfacción del cliente. Los clientes estuvieron más satisfechos debido a que el equipo produjo lo que el cliente realmente deseaba. La cuarta hipótesis no se pudo demostrar.

Estado de ánimo del equipo se estudió mediante el resultado de una encuesta de adherencia Shodan. Debido a que no se pudo determinar de los encuestados quiénes pertenecieron a ambos releases no se pudo demostrar la hipótesis aunque los equipos de indicaron conformidad y satisfacción en la forma de trabajo mediante XP.

Como conclusión se indicó entonces que mediante XP se pudo mejorar la productividad de los programadores y la calidad del producto.

3.2.8.2 Métodos Crystal21

Los métodos Crystal fueron desarrollados por Alistair Cockburn a comienzos de la década de los noventa. El creía que uno de los mayores obstáculos en el desarrollo de los productos era la escasa comunicación y luego modeló los métodos Crystal basándose en esas necesidades. A su vez tuvo en cuenta aspectos relacionados con las personas y el equipo de desarrollo como ser: colaboración, nacionalidad, y cooperación y la reducción de cantidad de artefactos producidos.

[21] www.crystalmethodologies.org

El nombre Crystal hace referencia a varias facetas de un mismo centro subyacente. El centro subyacente representa a los valores y principios mientras que cada faceta representa a un conjunto específico de elementos como ser: técnicas, roles, herramientas y estándares [AGILE-ECO].

Se deben invertir esfuerzos en mejorar las habilidades y destrezas del equipo de desarrollo, así como tener políticas de trabajo en equipo definidas que dependerán del tamaño del equipo. Por ello es que se establece una clasificación por colores, por ejemplo Crystal Clear de 3 a 8 miembros, y Crystal Orange de 20 a 50 miembros [AGILE-ISSI].

Las características fundamentales en los métodos Crystal son [AGILE-REQ]:

- Entregas frecuentes.
- Comunicación osmótica: significa que generalmente las comunicaciones intra-equipo son informales y no se dan en las reuniones pactadas.
- Perfeccionamiento reflexivo: permite al equipo evaluar el proceso de desarrollo, modificarlo y ajustar los cambios de acuerdo a las lecciones aprendidas de iteraciones anteriores.

Los métodos Crystal tienen tres prioridades básicas: seguridad en los resultados, eficiencia y habitabilidad de forma que los desarrolladores pueden vivir con Crystal [NEWM-FOW].

Los roles fundamentales en Crystal son: el de programador y el de diseñador semi-senior. Luego se requiere un representante, programador-diseñador y el usuario con una participación mínima de medio tiempo [ICONIX-PROC].

En [AGILE-ACOCK] se mencionan las políticas estándares de Crystal:

- El software se entregará de forma incremental y regularmente, cada 2 a 3 meses.
- El progreso es seguido mediante hitos que consisten en la entrega de software o decisiones importantes en lugar de documentos escritos.
- Existen pruebas automatizadas de regresión de funcionalidades de la aplicación.
- El usuario está involucrado directamente con el proyecto.
- Hay dos revisiones por versión.
- Las actividades de bajadas comienzan no bien logren ser estables como para su revisión.
- Los workshop de productos y ajustes de metodología serán mantenidos en el comienzo y a mitad de cada incremento.

Además, Cockburn indica los factores para incrementar la agilidad en el desarrollo de software:

- De dos a ocho personas por cuarto permite una comunicación más efectiva.
- Expertos on-site permiten obtener feedback y conocer sus necesidades rápidamente.

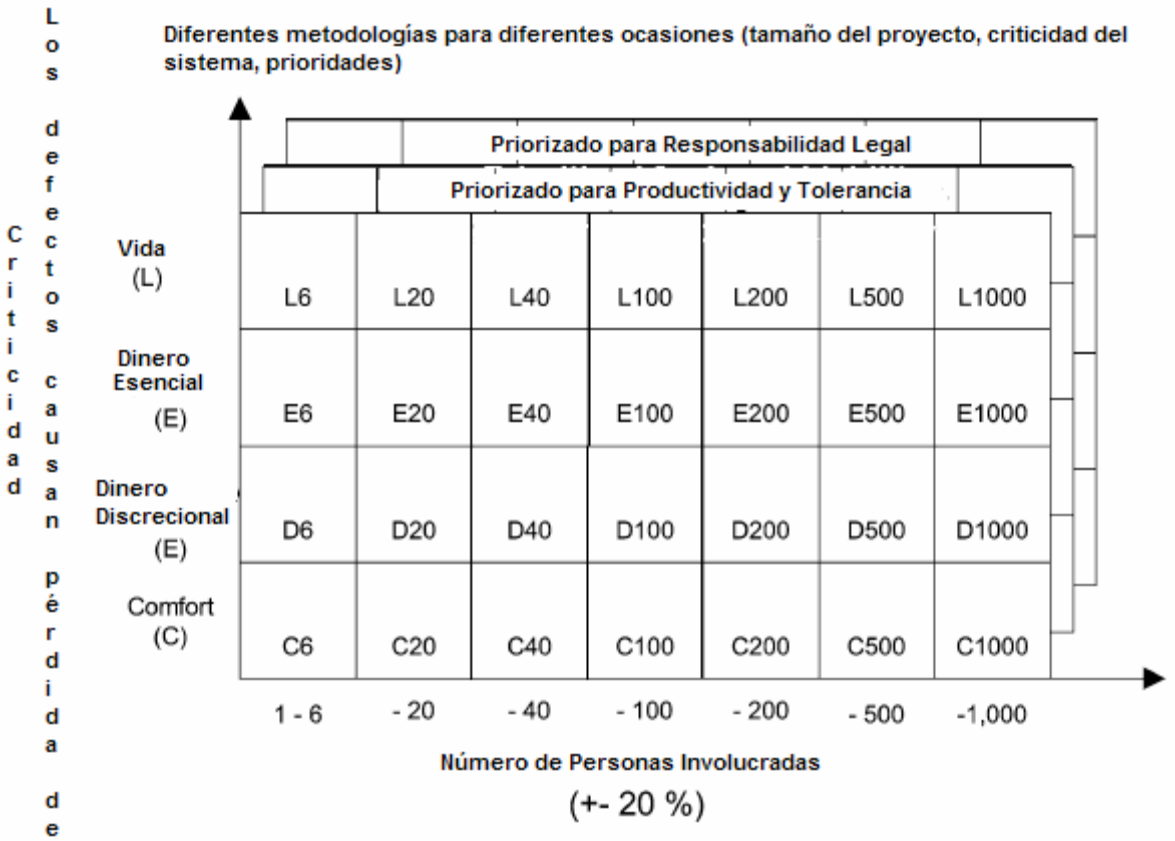
- Incrementos de uno a tres meses permiten a los desarrolladores reparar procesos y requerimientos.
- Los test de regresión automáticos mejoran la calidad del sistema.
- Por otra parte, considera que las metodologías deben ser auto-adaptables y que debido a la diferencia entre proyectos una actividad en uno puede incrementar la eficiencia pero en otro quizás desencadena el efecto contrario. La idea de que las metodologías sean adaptables lo llevó a pensar en la familia de metodologías Crystal.

Los miembros de las familias Crystal comparten las siguientes características:

- Están orientadas a la gente y a la comunicación, es decir que las herramientas, procesos y artefactos son soporte a las personas.
- Son altamente versátiles lo que significa que los equipos de trabajo no están obligados a trabajar con determinados procesos, sino que se los pueden seleccionar.

Las metodologías Crystal, se crearon de forma tal que san ajustables a diferentes tipos de proyectos. Estas se muestran en una matriz bidimensional. En el eje x se mide el tamaño del proyecto (de 1 a 6 personas, 7 a 20, 21 a 40, etc.) y en el eje y la criticidad del sistema (*Life* pérdida de vida teniendo en cuenta si ocurre un problema en el sistema, Essential Money cuando la pérdida de dinero esencial provoca bancarrota, Discretionary Money indicando que podría perderse dinero si falla el sistema y *Confort* cuando por problemas de sistema no se genere satisfacción en el usuario). La matriz está coloreada indicando que un proyecto se torna más “pesado” requiriendo coordinación de más personas y cumplir con más requerimientos críticos. Tal como se indica en [CASES-AGILESD], un proyecto crítico de la NASA, para guiar a un trasbordador con cinco personas asignadas, tendrá la clasificación L6.

A continuación, se incluye la matriz que representa lo indicado anteriormente.



Crystal Framework Method – Obtenido de www.crystalmethodologies.org

Los artefactos a producir son: secuencia de versiones, planificación de entregas, casos de uso, diagramas de diseño, notas, capturas de pantallas, modelo de objetos, código fuente, código para el traspaso de ambientes o migraciones, casos de prueba y manuales de usuario. Sin embargo para el caso de Crystal Clear Orange orientado a grandes proyectos, se incluyen más roles y la entrega de artefactos formales, reportes, etc.

En [CASES-AGILESD] se mencionan algunas ventajas de éstos métodos:

- Los métodos Crystal son adaptables, por lo tanto se ajustan a las características propias de cada proyecto, como ser su nivel de criticidad, tamaño, etc.
- Crystal alienta la comunicación frecuente, rica e informal.
- Permite la entrega de versiones de forma rápida.
- Considera a las personas, su interacción, comunidad, perfiles, talentos y comunicación.

3.2.8.3 Scrum[22]

Es una metodología de desarrollo de software que fue formalizada por Ken Schwaber en 1996, pero aplicado por Jeff Sutherland. Luego Sutherland y Schwaber la refinaron y extendieron.

Es un proceso que acepta que el análisis, diseño y el proceso de desarrollo son impredecibles y tuvo éxito con vendedores de software independientes. Se emplea un mecanismo de control para manejar lo impredecible y controlar los riesgos. Los resultados son la flexibilidad, responsabilidad y la fiabilidad [SCRUM-SCHW].

"Scrum" proviene del juego Rugby. Un scrum ocurre cuando los jugadores de cada equipo se entrelazan unos a otros con el objetivo de tomar ventaja en el campo de juego [AGIL-DACS].

No está concebido como un método independiente, más bien es un complemento para otras metodologías como ser: RUP, XP, etc.

Es un proceso adaptativo de gestión de proyectos, auto-organizado y con pocos tiempos muertos. Soporta el desarrollo iterativo e incremental.

Una vez completado el planeamiento inicial del proyecto una serie de fases llamadas sprints, permiten entregar el producto de forma incremental. Estos sprints dividen al proyecto en pequeñas fases formando iteraciones que pueden durar un mes aproximadamente.

El equipo confecciona una lista organizada y con prioridades, que se denomina Backlog, que contiene todas las tareas identificadas anteriormente y que luego oficiarán como guía del equipo. Para aquellas tareas que se considere insumirán demasiado tiempo, el equipo tratará de dividirlas en subtareas.

Este documento se mantendrá a lo largo de todo el proyecto incorporando constantemente actualizaciones.

Luego de cada sprint, el equipo actualizará este listado, repriorizará las tareas, seleccionará alguna de ellas y ejecutará un sprint. Todos los miembros del equipo deben estar de acuerdo en completar las tareas que crean realizables durante cada sprint [SCRUM-STEAM].

Durante el sprint no se puede modificar lo acordado en el Backlog excepto en ocasiones, donde no es viable continuar ya sea debido a que la tecnología pactada no funciona, cambió el negocio o el equipo tuvo interferencias.

Durante cada sprint todo el equipo mantendrá reuniones periódicas. Esta es una forma de que todo el equipo se sienta parte del proyecto y cada uno pueda hacer visible su trabajo al resto del equipo.

22 www.controlchaos.com

Los equipos están formados normalmente entre 5 a 10 personas entre los cuales se encuentran diferentes perfiles, como ser: programadores, diseñadores, analistas, testers, etc. Solo se permitirán cambios dentro de los equipos entre sprints.

Por su parte, Scrum se basa en dos pilares: adaptabilidad y team empowerment:

Team Empowerment, se refiere a que los equipos sean relativamente autónomos, pudiendo ellos elegir la forma de realizar el trabajo asignado entre cada incremento, eliminar las ineficiencias que restringen a los equipos y mejorar la productividad. De esta forma se reduce la burocracia de la administración de los equipos de trabajo [AGILE-T&P].

Para Schwaber observar la interacción de los equipos es la mejor forma de entender la compleja interacción de las personas, tecnología, necesidades, intereses, y satisfacción.

Por otra parte, con las reuniones diarias se logra la visibilidad del progreso del proyecto. Con la transparencia que se logra se evita la pérdida de tiempo en políticas y con la independencia de los equipos, éstos se tornan auto-organizados y auto-regulados.

La **adaptabilidad** se refiere al equilibrio que el equipo mantiene durante cada incremento dado que es aislado de disturbios. Al finalizar un sprint, los incrementos son puntuados mediante una reunión en la cual se revisa la iteración y se planifica lo que será tratado en la siguiente iteración [AGILE-T&P].

Características de Scrum tomadas de [SCRUM-SCW]:

- En los ambientes de trabajo abiertos se desataca el feedback y se reconocen los errores. A su vez, se espera que todos los miembros de un equipo entiendan cada problema y todos los pasos en el desarrollo del sistema para resolverlo.
- La primera y última fase (Planeamiento y Cierre) consisten en procesos bien definidos. El cómo se hacen las cosas es implícito. El flujo es lineal, con algunas iteraciones en la fase de planeamiento.
- La fase Sprint es un proceso empírico. Muchos de esos procesos en esta fase son indefinidos o sin control, por ello se los trata como una caja negra que requiere controles externos. Luego, los controles, incluyendo la administración de riesgos, son puestos en cada iteración de la fase Sprint para evitar caos mientras se maximiza la flexibilidad.
- Los Sprints son no lineales y flexibles. Si está disponible, se usa el conocimiento de los procesos, en caso contrario el conocimiento tácito, las pruebas y el error son usados para construir conocimiento sobre esos procesos. Los Sprints se usan para desarrollar el producto final.
- El proyecto permanece abierto hasta la fase de cierre. Las entregas pueden ser modificadas en cualquier momento durante la fase de planeamiento y las fases de Sprint.
- Las entregas son determinadas durante el proyecto y basadas en el entorno.

- Con cada nueva iteración, se revisa el trabajo pendiente y se seleccionan como incremento las nuevas funcionalidades.
- Al finalizar cada iteración, el equipo presenta el incremento de la funcionalidad a los implicados en el proyecto.

Roles

A continuación se detallan los roles y responsabilidades presentes dentro de Scrum y que hacen en conjunto posible el desarrollo del producto citado por Scrum Alliance[23].

Product Owner:

- Definir las características del producto.
- Decidir sobre la fecha de versión y el contenido.
- Responsabilizarse de la ganancia del producto.
- Priorizar las características según el valor de mercado.
- Ajustar las características y prioridades cada 30 días según se necesite.
- Aceptar o rechazar el resultado del trabajo.

Scrum Master:

- Asegurarse de que el equipo sea funcional y productivo.
- Permitir la cooperación estrecha entre todos los roles y funciones.
- Quitar barreras.
- Evitar que el equipo tenga interferencias del exterior.
- Asegurar que el proceso sea fluido incluyendo la realización de invitaciones a scrum diario.
- Revisiones de sprint.
- Reuniones de planeamiento de sprint.

Equipo de Desarrollo:

- Compuesto de diversas funciones entre 2 a 7 miembros.
- Selecciona objetivos del sprint y especifica resultados del trabajo.
- Tiene el derecho de realizar todo dentro de los límites de las directivas del proyecto para poder así alcanzar el objetivo del sprint.
- Se organiza a si mismo y su trabajo.
- Realiza demos del trabajo al Product Owner.

[23] www.scrumalliance.org/view/scrum_roles

Fases

Scrum se describe en las fases que se describen a continuación. Para finalizar se incluye un diagrama que lo representa gráficamente.

Pre-Juego:

Se compone del planeamiento y la arquitectura ó diseño de alto nivel.

1) El *Planeamiento* incluye la definición del sistema que se está construyendo. Se define la nueva versión basada en el backlog (trabajo atrasado), y en los requerimientos conocidos. Se estiman los esfuerzos necesarios para cada requerimiento. Este backlog se irá modificando en tanto y en cuanto surjan nuevos ítems o se posean estimaciones más precisas.

La planificación puede incluir la definición del equipo de trabajo, herramientas y recursos, cálculo de riesgos, entrenamientos, etc. [AGILE-METH].

2) La *Arquitectura* incluye el diseño de alto nivel de acuerdo con lo presente en el backlog. Si hubiese cambios se incluyen los problemas que éstos podrían causar.

Desarrollo ó Juego:

Se efectúa el desarrollo con nuevas funcionalidades que incluye constantes con respecto a variables de tiempo, requerimientos, calidad, costos, y competencia. La interacción con estas variables define el fin de esta fase. Existen múltiples sprints de desarrollo iterativo, o ciclos en el desarrollo del sistema [SCRUM-SCHW]. Cada uno incluye las fases tradicionales de desarrollo de software: requerimientos, análisis, diseño, evolución y entrega. Se planifica que cada sprint durará entre una semana y un mes [AGILE-METH].

Post-juego:

Representa el cierre de la versión e incluye la documentación final, el conjunto de pruebas y la versión en si. Se entra en esta fase cuando se han aprobado las variables de entorno como es la completitud de los requerimientos. En esta instancia no habrá más ítems o tareas ni otros nuevos requerimientos [AGILE-METH].

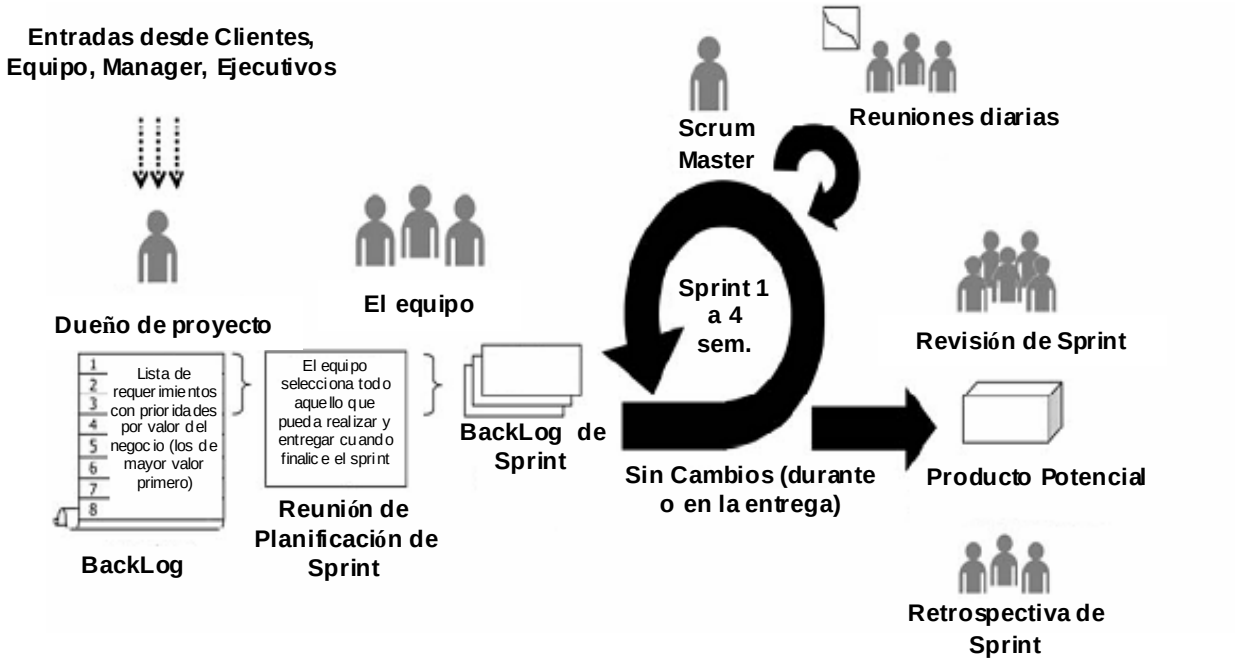


Diagrama de Scrum. Fuente: [SCRUM-PRIM] 1

Características que se compararán obtenido de [AGILE DACS]:

Característica	Detalle
Tamaño del equipo	Los equipos están conformados por menos de siete personas. Deben al menos incluir un Desarrollador, un Ingeniero en Aseguramiento de Calidad y un Analista.
Duración de la Iteración	Schwaber sugirió originalmente sprints de 1 a 6 semanas, pero normalmente abarcan 4 semanas.
Soporte para equipos distribuidos.	Si bien no hace explícitamente mención a esto, un proyecto consiste de múltiples equipos que puedan ser fácilmente distribuidos.
Criticidad del sistema	No define un comportamiento ante esto.

Ventajas:

Sutherland estableció que en todos los casos que estudió, los resultados demostraron que Scrum mejoró la comunicación y la entrega de código funcionando. Además demostró ser escalable en los grandes proyectos de programación y ser conveniente para las organizaciones con ambientes diferentes [AGILE-T&P].

[SCRUM-SCHW] también menciona las ventajas del uso de Scrum:

- Está diseñado para proveer mecanismos de control para la planificación de una versión de producto y luego administrar variables a medida que el proyecto avance. Esto

permite a las organizaciones cambiar las entregas del proyecto en cualquier momento, entregando la versión más apropiada.

- Permite que los desarrolladores propongan soluciones durante el avance del proyecto, es decir a media que se va aprendiendo y se producen cambios en el ambiente y no de forma apresurada el inicio del mismo.
- Los pequeños y colaborativos equipos de desarrolladores son capaces de compartir conocimiento sobre el desarrollo del proyecto.

Además se considera como ventaja su enfoque a la entrega rápida del sistema, libertad de trabajo en los equipos, el feedback continuo y la mejora en el proceso de forma continua.

Es ideal para proyectos críticos y complejos, en los que la calidad y velocidad de implementación son fundamentales y los requerimientos son vagos o muy cambiantes.

Limitaciones

Una de las limitaciones de Scrum es que cada miembro de los equipos de desarrollo debe entender cada problema y los pasos necesarios a desarrollar para lograr su solución.

Además debe complementarse con otras metodologías como ser XP para no caer en procesos caóticos y si no se realiza un seguimiento se puede caer en un modelo de codificar y probar.

En [AGILE-REQ] se mencionan algunas desventajas en Scrum:

- No especifica una técnica de elicitación de requerimientos no funcionales. Debería designarse una parte del backlog para la especificación de éste tipo de requerimientos.
- No maneja el concepto de trazabilidad de requerimientos y si bien la justificación de Schwaber es que los requerimientos no cambian a través de un proyecto ágil sino que se desarrollan, dado que el valor está puesto en el negocio, si se falla al implementar algunos requerimientos, se disminuye el valor del negocio. Por ello es recomendable incluir manejo de requerimientos.
- Existe el riesgo de no implementar requerimientos de baja prioridad, debido a que los equipos de desarrollo solo trabajan con los ítems de alta prioridad. Hay que recordar que éstos se encuentran siempre debajo, en la pila de los ítems y que podría ser que nunca se los llegue a implementar.

Ejemplos

La idea de la construcción de un equipo auto-organizado como en Scrum fue exitoso en empresas como Honda, Canon y Fujitsu, y en equipos promovidos por el Profesor Peter

Senge del MIT en su trabajo “The Fifth Discipline: the Art and Practice of the Learning Organization Currency” de 1990.

En el trabajo [SCRUM-PRIM] se pueden observar Resultados de la puesta en práctica de Scrum, se muestran resultados positivos de una encuesta incluyendo ítems como ser: Adaptabilidad, Productividad, Colaboración y Cooperación, etc.

En el sitio oficial de Scrum[24] se pueden encontrar casos de estudio de aplicaciones SCRUM.

3.2.8.4 Dynamic Systems Development Method[25]

Es un Método de Desarrollo de Sistemas Dinámicos (DSDM) originado en un Consorcio en 1994 convirtiéndose en un framework de desarrollo rápido de aplicaciones (RAD) en 1997 en Reino Unido.

El objetivo era lograr una metodología independiente de las herramientas.

Los desarrolladores del método sostienen que por servir de método, provee un framework de controles para RAD, suplementado con la guía de cómo usar eficientemente estos controles. Su idea principal consiste en ajustar en primera instancia tiempo y recursos y luego ajustar la cantidad de funcionalidades a efectuar según ello y no viceversa [AGILE-METH].

Permite que las funcionalidades del sistema varíen sobre la vida del proyecto como nuevas cosas aprendidas. Sin embargo el control es mantenido por medio del uso de cajas de tiempo [AGILE-QES]. Utiliza el prototipado en lugar de mantener mucha documentación.

Esta metodología se basa en nueve principios [DSDM-PRINC]:

1) Es imperativo que el usuario se involucre dado que reduce el error en la percepción del usuario y debido a esto reduce los errores.
2) Los equipos deben poder tener facultad de tomar decisiones dado que en caso contrario los cambios disminuirían la velocidad del proyecto de forma significativa.
3) El foco principal está en la entrega de productos con frecuencia para detectar de forma temprana los errores.
4) El criterio esencial de aceptación de la entrega de versiones del producto es la conformidad con el negocio.

[24] www.controlchaos.com/
[25] www.dsdm.org

5) Es obligatorio el desarrollo iterativo e incremental descomponiendo el proyecto en paquetes pequeños de funcionalidades, agregando en cada versión nuevos requerimientos hasta alcanzar la completitud del sistema.
6) Todos los cambios durante el desarrollo deben ser reversibles dado que teniendo en cuenta que se hacen pequeños incrementos, si se vuelve atrás un desarrollo no sería tan grave la pérdida del trabajo.
7) Los requerimientos son especificados solo en un alto nivel.
8) Las pruebas son integradas a través del ciclo de vida.
9) Enfoque colaborativo y cooperativo dado que son características esenciales para el éxito del proyecto.

Fases según [AGILE-METH] y [AGILE-QES]

Es un proceso que consta de cinco fases, las dos primeras son secuenciales y las demás iterativas e incrementales. El uso de tres modelos iterativos e interactivos y cajas de tiempo pueden ser usados para construir planes de proyectos flexibles [AGILE-QES].

Una caja de tiempo dura un período predeterminado de tiempo (entre unos días y unas semanas) y una iteración debe terminar entre cajas de tiempo.

Estudio de factibilidad

Esta fase dura unas pocas semanas.

De acuerdo al tipo de proyecto, tareas del la organización y su gente, se toma la decisión de usar o no DSDM. Se centra en el estudio de factibilidad teniendo en cuenta las posibilidades técnicas y los riesgos en ella. Se prepara un reporte de factibilidad, un plan de desarrollo y opcionalmente un prototipo si el negocio o la tecnología son desconocidas como para decidir si proceder en la siguiente fase o no [AGILE-METH].

Estudio del negocio conforman una fase.

Se analizan las características esenciales del negocio y la tecnología. Los procesos de negocio y las clases de usuario son descriptos en la Definición de Área de Negocio. Además se genera la Definición de Arquitectura del Sistema y Planes de Prototipos.

Iteración de Modelo Funcional

Es un proceso de acumulación y prototipado de requerimientos basados en una lista inicial de requerimientos en orden de prioridades. Los requerimientos no funcionales son a menudo también especificados en esta fase.

Los prototipos se mejoran de forma tal que alcancen a ser el sistema final, prototipado no desechable. El Modelo Funcional es producido como salida de esta fase y contiene el código del prototipo y los modelos de análisis. Las pruebas son continuas y conforman una parte fundamental de esta fase.

Se crea una lista de funcionalidades con prioridades asignadas, documentos del prototipo funcional, que contienen comentarios de los usuarios en cada incremento, un listado de requerimientos no funcionales, un análisis de riesgos en desarrollos futuros.

Iteración de Diseño y Construcción

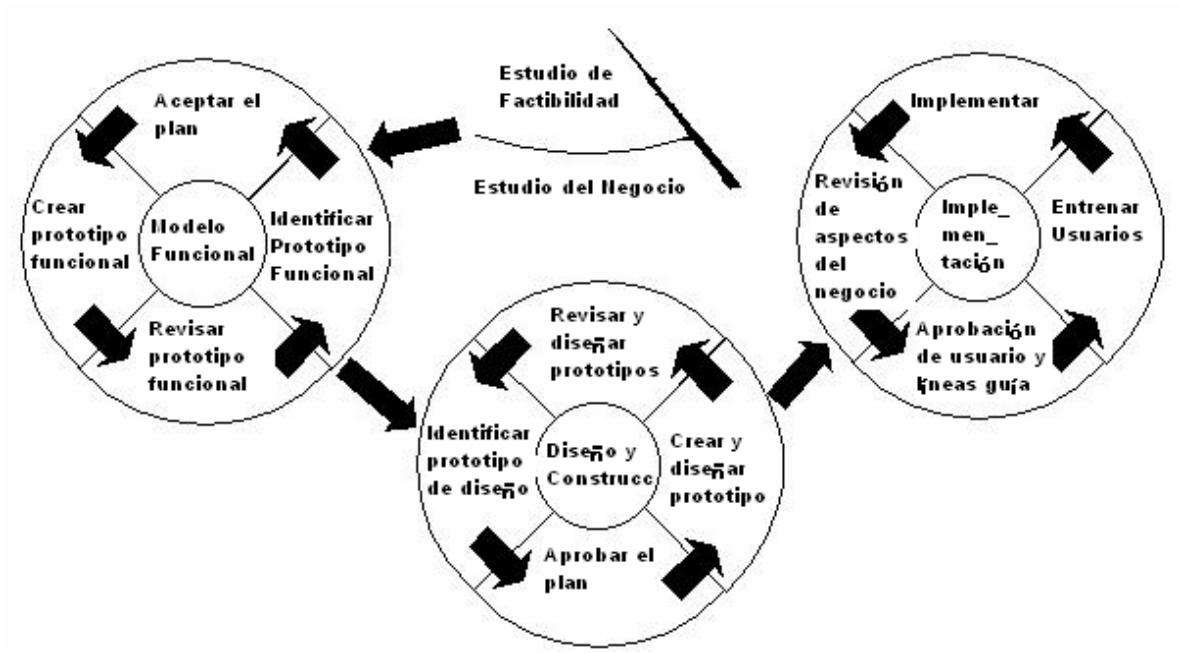
Básicamente es donde se construye el sistema que cumple con los requisitos básicos.

Refina el prototipo para conocer todos los requerimientos, tanto funcionales como no funcionales y la ingeniería de sistemas para conocer todos los requerimientos. Durante una caja de tiempo pueden darse un conjunto de funciones de negocio entre la fase anterior y ésta y luego otro conjunto puede ir a través del mismo proceso en una segunda caja de tiempo.

Implementación

Despliega el sistema en el ambiente de usuario. Se entrega un manual de usuario y un reporte de Revisión del Proyecto que resume el proyecto y de acuerdo a los resultados que se obtengan se incluirá detalle de futuros desarrollos.

A continuación se incluye un diagrama que resume lo anterior.



Ciclo de vida DSDM. Obtenido de: http://www.dsdm.org

Debido a que la naturaleza de DSDM es de framework, no especifica la cantidad de personas dentro de un equipo, la cantidad de iteraciones, distribución o criticidad de un sistema [AGIL-DACS].

Ventajas.

DSDM no requiere que los usuarios implementen toda la estructura del proyecto, solo demanda que se respeten los nueve principios. Cualquier administrador de proyecto puede implementar el proceso de desarrollo resultando un proceso ágil en mayor o menor medida dependiendo de la situación y restricciones, se puede combinar DSDM con otras metodologías y en diversos ambientes. Provee técnicas para manejar dinámicamente las planificaciones, dado que se basa en el principio de que "ningún plan sobrevive al primer día de desarrollo" [DSDM-ZURICH].

Limitaciones

En el estudio [AGILE-REQ] se mencionan algunas desventajas de DSDM, como ser:

- El tiempo de prototipado es grande, si bien es considerado uno de los factores críticos de éxito.
- Cuando es aplicado a proyectos grandes y distribuidos el análisis de requerimientos puede tornarse complicado y puede llegarse a un punto muerto y en caso en que la elicitación de requerimientos se haga en sesiones grupales no hay garantía de que el equipo de trabajo conozca todos los requerimientos necesarios resultando un diseño defectuoso.
- No existe una explicación explícita para la trazabilidad dentro de un proyecto en el análisis de impacto referente al cambio de requerimientos o ante nuevos.

Por otra parte, debido a que DSDM no resuelve todos los problemas de desarrollo de software, cuando los nueve principios no puedan ser implementados DSDM no será la mejor opción. A su vez, una de las debilidades de DSDM es que es difícil, lento y costoso de comenzar su implementación (por los costos de licencias), requiere un cambio cultural en la organización dado que rápidamente se cambian las entregas por tareas [DSDM-ZURICH].

Ejemplo de implementación

En [DSDM-PRINC] se mencionan compañías que han aplicado DSDM para el desarrollo de software: Shell, Loyds Bank Insurance Services, British Telecom, British Airways, Deutsche Bahn, Hewlett-Packard, Renault, organización de desarrollo offshore en India para en Xansa, etc.

3.2.8.5 Lean Programming[26]

Comenzó con Bob Charette, presidente de ITABHI de la industria automotriz en 1980, quién pensó que para ser realmente ágil se necesitaba ver cómo las compañías efectuaban los cambios de forma top-down [AGIL-DACS]. Luego tuvo éxito en varias compañías de telecomunicaciones europeas.

Mary Poppendieck introdujo principios al desarrollo de software basándose en su experiencia sobre Lean Manufacturing y Total Quality Management (TQM) que empleó para mejorar la producción. Debido a que los principios estaban pensados para otro contexto, efectuó determinadas modificaciones sobre ellos de forma tal que sean aplicables al desarrollo de software.

Los siete principios (extraídos de [LEAN-POPP] y [27]) son:

- *Eliminar desperdicios*. Tiene como objetivo observar toda la línea de tiempo desde que se efectúa un requerimiento hasta que se entrega el producto de software buscando todos los agregados que no aportan valor y eliminándolos. Los desperdicios son aquellas cosas que interfieren en el brindar a los clientes lo que valoran en el tiempo y espacio, donde proveerán en mayor valor. Por ejemplo cualquier funcionalidad ó código adicional que retrasa el proceso de desarrollo de software, requerimientos que no son claros, burocracia y manejos que no producen valor y comunicación interna débil. Una vez que se identificó aquello que no aporta valor, debe ser tratado de forma iterativa hasta que sea totalmente removido. Básicamente consiste en realizar refactoring, es decir, reconstrucción o mejoras que durante el transcurso de las iteraciones harán que el sistema funcione mejor.
- *Construir calidad*. Se propone mediante una organización disciplinada, construir con calidad desde el inicio de la codificación, introduciendo pruebas tan pronto como sea posible y realizando inspecciones antes de que los defectos ocurran. Se emplean iteraciones de corta duración en las que se efectúan modificaciones y pruebas de integración. Esto permite obtener feedback asiduamente permitiendo hacer ajustes y mejoras en las siguientes iteraciones. Este punto es cuestionable, debido a que plantea como medida de construcción de calidad a las inspecciones antes de que los defectos ocurran. El punto que aquí no queda claro es: si los defectos no ocurren debido a la existencia de las inspecciones, o a que desde un principio se propone no generar fallas. Ahora bien, en el caso de que las fallas ocurran a posteriori entonces uno podría preguntarse, para qué sirvieron dichas inspecciones.
- *Crear conocimiento*. Los procesos de desarrollo deben alentar el aprendizaje sistemático y la mejora de los procesos a través del ciclo de vida de desarrollo, debido a que

[26] www.poppendieck.com
[27] http://en.wikipedia.org/wiki/Lean_software_development

en ambientes complejos, siempre hay problemas para los que se buscará su causa y se experimentarán soluciones hasta encontrar la mejor. Los equipos de desarrollo tienen la responsabilidad de la mejora de los procesos.

- Se requiere también eliminar las especulaciones que nos llevan a tomar decisiones dado que los mejores resultados se obtienen cuando se basan en hechos y no en predicciones. Entonces lo mejor es tener la capacidad de poder esperar a que sucedan los hechos para luego actuar rápidamente y de forma correcta. *Esta concepción es cuestionable, pareciera malinterpretarse el concepto de especulación con predicción. Se puede especular de acuerdo a lo que la experiencia le haya enseñado a uno, pero también se puede predecir un 100% de probabilidad de acierto, en base a teorías generales. Con el objetivo de ejemplificar, se puede predecir que una estrella de más de cien veces la masa del sol, no vivirá más de 100 millones de años y se convertirá en un agujero negro de acuerdo a la Teoría General de Einstein*.
- *Decidir lo más tarde posible.* Se debe tratar de tomar decisiones reversibles de forma tal que luego sea posible efectuar cambios fácilmente sobre todo cuando se desarrollan las primeras funcionalidades del sistema. Lo ideal posponer la toma de decisiones hasta que se tengan los hechos suficientes y uno no tenga que basarse en predicciones.
- *Efectuar entregas tan pronto como sea posible* favorece la obtención de feedback de los clientes y la incorporación de defectos o mejoras en las siguientes iteraciones. Si las iteraciones son cortas entonces el aprendizaje y la comunicación entre los equipos es mejor.
- *Potenciar el equipo*. Esto significa que la opinión de los desarrolladores sea escuchada, que las personas no sean tratadas como un recurso. Se debe reforzar el equipo proveyéndole capacitación, motivación, propósitos de trabajo, deben tener soporte de su líder quien les ayude a resolver situaciones problemáticas.
- *Optimizar el todo* desde el momento en que se tiene una necesidad del cliente hasta que el software es entregado. *Aquí hay que tener en cuenta que muchas veces es conveniente optimizar, en un programa, solo aquellas rutinas que más se utilizan. Por ejemplo, cuál sería el objetivo de optimizar el 100% del código de un programa, si en realidad el programa se pasa la mayor cantidad del tiempo ejecutando una rutina que representa un pequeño porcentaje del código total. Lo ideal es optimizar ésta rutina y se conseguirá mayor performance.*

Por otra parte, en Lean los cambios se consideran riesgos pero si se manejan de forma adecuada pueden convertirse en oportunidades que permitan mejorar la productividad del cliente [AGILE-ISSI].

Las estrategias de Lean se centran en el manejo de estrategias según Highsmith:

1) La principal prioridad es satisfacer a los clientes.
2) El éxito depende de la activa participación del cliente.
3) Proveer siempre el mejor valor de acuerdo al dinero invertido.
4) Cada proyecto LD es un esfuerzo del equipo.
5) Todo se puede cambiar.
6) El foco en el dominio y no en las soluciones.
7) Complete, no construya.
8) Un 80% de la solución hoy en lugar del 100% mañana.
9) Es esencial el minimalismo.
10) Las necesidades determinan la tecnología.
11) El producto crece si las funcionalidades crecen.
12) No extralimite el desarrollo LEAN.

Es una pieza operacional en el enfoque de tres capas que lleva a efectuar cambios tolerantes en el negocio. Provee un mecanismo de entrega para implementar riesgos empresariales.

Charette indicó que las empresas debían ser organizaciones tolerantes a los cambios de forma que puedan imponer el cambio en los competidores. Además, dejó tres mensajes claves para los desarrollares ágiles y participantes en la tecnología de la información. Primero: la adopción de ASDE (ecosistema de desarrollo de software ágil) requerirá ventas estratégicas en niveles altos dentro de una organización, segundo: el mensaje estratégico que venderá ASDEs es la habilidad de arrancar oportunidades desde situaciones de gran movimiento hasta las más riesgosas, y tercero: los proponentes de ASDEs deben entender y comunicar a sus clientes los riesgos asociados con los modelos ágiles y luego, las situaciones en las cuales están y aquellas que no son las apropiadas. Indicó también que estos tres objetivos necesitan ser llevados a cabo de forma concurrente [AGILE-QS].

Ventajas

Aporta principios para agilizar el desarrollo del software.

Según lo indica [AGILE-ECO] la contribución de Bob Charette sobre el movimiento de desarrollo de software ágil se resume a cuatro partes: foco estratégico, vinculación a la producción lean, riesgo empresarial, y extensión de objetivos. Mientras que XP se enfoca en el desarrollo a nivel cliente y Scrum y DSDM en la administración del proyecto, Lean Development vende en el nivel ejecutivo más alto.

Extiende el concepto del cambio en las metodologías tradicionales donde lo considera como un riesgo de pérdida a ser controlado con prácticas restrictivas de management a una

postura diferente, la de ver a los cambios como motores de oportunidades a ser seguidos usando el riesgo empresarial.

El desafío requiere un repensamiento radical acerca de cómo se desarrolla el software. El desafío es a nivel management y ejecutivo tipo top-bottom. Plantea la necesidad de administración, al nivel más alto, para comprender mejor la administración de los riesgos de la empresa. El riesgo de la exploración debe ser aceptado a ese nivel y no llevado a niveles inferiores a los equipos de proyecto como objetivos imposibles.

Limitaciones

Al igual que DSDM, como LD es más una filosofía que un proceso de desarrollo, no define el tamaño de equipo, la duración de las iteraciones, la distribución del equipo ni la criticidad del sistema [AGIL-DACS].

Si bien se han expresado ventajas, este método no especifica claramente cómo lograr estos resultados.

Ejemplo de implementación

Solo se han encontrado menciones de implementaciones en empresas de telecomunicaciones, las mismas carecen de argumentos suficientes para poder obtener conclusiones contundentes respecto a las ventajas y desventajas producto de su aplicación.

3.2.8.6 Feature-Driven Development

Es un proceso ideado por Jeff De Luca, director de proyecto de Nebulon[28] una consultora de IT en Melbourne Australia. Participó en un proyecto que durante dos años recaudó mucha documentación y nada de código. Este proyecto que abarcaba un gran dominio de negocio, quedó declarado como de imposible creación. A partir de eso se dedicó a proyectos livianos en el desarrollo.

Peter Coad con experiencia en el paradigma de Orientación a Objetos y el desarrollo conducido por funcionalidades, se unió a De Luca en lo que luego se denominó FDD (Feature-Driven Development) cuyos primeros proyectos ya mostraban al cliente en poco tiempo porciones del sistema. Se insumía un mes aproximadamente en el modelado y luego otras semanas en la descomposición de funcionalidades y en el planeamiento de iteraciones

[28] www.nebulon.com/fdd, www.coad.com/peter/#fdd

cortas. Luego se construyeron pruebas de concepto y a partir de allí se logró con equipos de 50 personas entregar funcionalidades en 15 meses [AGILE-QS].

Según De Luca, lo fundamental es tener en el proyecto buenos profesionales, con alto nivel técnico, talentosos, expertos en el dominio, desarrolladores, programadores, etc.

FDD es un método que consiste en cinco procesos: se comienza con el dominio y el equipo de desarrollo trabajando juntos con un modelador experimentado en componentes u objetos y construyen una lista detallada con las funcionalidades. El modelo se ajustará a lo largo del camino. El equipo de desarrollo asigna prioridades e identifica cuáles deben incluirse en el producto. Luego, las mismas son incluidas en un plan de alto nivel. En iteraciones de dos semanas se efectúa el diseño por actividad seguido de su construcción. Esto promueve el desarrollo concurrente entre cada incremento. Las iteraciones incluyen: inspecciones de diseño, pruebas unitarias, integración e inspección del código antes de que el programador acepte las funcionalidades promovidas en la construcción principal [FDD-PALMER].

A continuación se incluye la descripción de cada una de las fases involucradas en el ciclo de vida de FDD y para finalizar un diagrama que las representa [CASES-AGILESD]:

- Desarrollo de un modelo General.

Los expertos en el dominio del problema junto a los desarrolladores generan un modelo esqueleto que luego será refinado y ampliado por equipos más pequeños asignados a investigar determinados requerimientos. En esta fase, se generan diagramas de clase, diagramas informales de secuencias, lista de funcionalidades y notas de modelos alternativos. El tiempo que se debe tomar para esta fase es de un 10% del tiempo total del proyecto más otro 4% sobre bases ya existentes.

- Construcción de una lista de funcionalidades.

En base a la lista de funcionalidades de la fase anterior, se arman conjuntos de funcionalidades. A cada una se le asignará una prioridad teniendo en cuenta las siguientes categorías: “se debe tener”, “sería mejor tenerlas”, “y si pudiéramos tenerlas”, “a futuro”.

El tiempo que debe insumirse en esta fase es de un 4% más otro 1% sobre bases ya existentes.

- Plan de versiones según las funcionalidades a implementar.

Se planifica cómo será la implementación de las funcionalidades y los tiempos insumidos. Se asignan responsabilidades para las clases y conjuntos de funcionalidades.

El tiempo que debe insumirse en esta fase es de un 2% más otro 2% sobre bases ya existentes.

- Diseño en base a las funcionalidades.

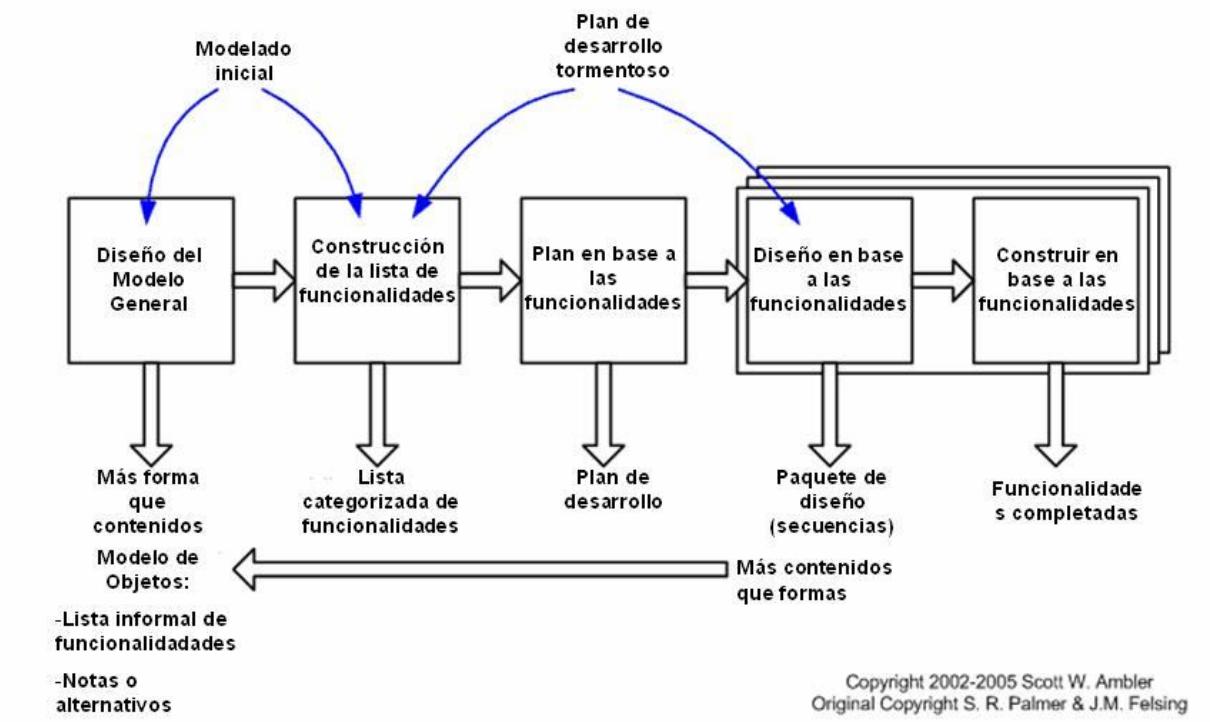
Las funcionalidades que deban ser implementadas serán analizadas con el fin de identificar las clases que contribuyen a la implementación. Los dueños de las clases identificadas son llamados para ayudar a compilar los diagramas de secuencias requeridos y la especificación de los métodos. Los equipos luego efectúan una inspección al diseño.

- Implementación.

Cada dueño de clase implementa los métodos asignados incluyendo las pruebas. Una vez que se inspecciona el código, los dueños de clase son habilitados para verificar cambios en el sistema de administración de configuraciones.

Las iteraciones de diseño y construcción por funcionalidad deben tomar el restante 77% del tiempo de proyecto. Cada una de éstas iteraciones no debe insumir más de dos semanas.

A continuación se incluye un diagrama que muestra el ciclo de vida de FDD.



Ciclo de vida de FDD

FDD se centra en la gente en lugar de la documentación.

Pone especial foco en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software [AGILE-ISSI].

Característica	Detalle
Tamaño del equipo	Fue diseñado para pequeños equipos pero puede ampliarse a grandes proyectos. Dentro de estos equipos se encuentra personal especializado en el dominio del negocio y en lo técnico.
Duración de la Iteración	Hasta dos semanas.
Soporte para equipos distribuidos.	Es diseñado para equipos múltiples y si bien no tiene soporte para equipos distribuidos, podría adaptarse a ellos.
Criticidad del sistema	No especifica

Ventajas.

En el estudio [CASES-AGILESD] se mencionan las prácticas recomendadas por FDD cuya aplicación trae las siguientes ventajas:

- Con el uso de equipos de funcionalidades trabajando en forma conjunta para producir una solución , se logra la selección de la solución más apropiada sobre un conjunto de soluciones posibles. Para esto, el tamaño de equipos recomendado es de tres a seis miembros por equipo.
- Las entregas frecuentes proveen la habilidad de demostrar un sistema funcionando lo más actualizado posible. Permite ejecutar pruebas, verificar si la integración a la versión actual es correcta o rectificar problemas de integración lo antes posible evitando llegar al punto en que las mismas se tornan inmanejables.
- Provee la habilidad de entregar un reporte de estado. Todos los programadores jefe se reúnen con el administrador de versiones una vez por semana. Cada jefe indica verbalmente el estado de avance de las funcionalidades que tiene asignadas. Esto sirve a los otros jefes para conocer el progreso de las otras funcionalidades. El administrador de versiones genera un reporte que luego puede ser mostrado a los desarrolladores.

Limitaciones.

En el estudio [CASES-AGILESD] se mencionan algunas limitaciones relacionadas con la implementación de ésta metodología, las mismas son listadas brevemente a continuación.

- Si bien las prácticas FDD no son nuevas, la combinación de las mismas si. El no adherirse a todas las prácticas pueden resultar en la inhabilidad de aplicar la metodología.
- El desarrollo guiado por funcionalidades permite a los desarrolladores enfocarse en la implementación solamente de aquellas que aportan más valor para el cliente y desde la

perspectiva del cliente, esta estrategia provee una medida de estatus del proyecto más significativa.

- Cuando existen dependencias entre clases pueden suceder que un desarrollador deba esperar a que otro efectúe determinada modificación antes de poder completar su propia funcionalidad.
- Potencial pérdida de conocimiento cuando un equipo sale del proyecto.
- El uso de equipos por funcionalidades que conforman una estructura dinámica de equipo que permite a los dueños de clase estar disponibles para múltiples equipos de desarrollo y para diferentes funcionalidades es una práctica para evitar que ocurran las dos situaciones negativas mencionadas anteriormente. Sin embargo, podrían darse casos, de acuerdo a las características propias de cada proyecto, en que se produzcan cuellos de botella y pérdida de conocimientos.
- No da soporte para el testing y deployment y no es el más indicado en pequeños proyectos donde no se tiene la suficiente cantidad de personas para cubrir los roles necesarios. En el caso citado se resolvió asignando dos roles a una misma persona asumiendo el riesgo que ello implicaba[29].

3.2.8.7 Iconix

ICONIX surgió antes de UML y Unified Process como una síntesis y búsqueda de buenas prácticas de las metodologías originales que formaron UML (técnica de modelado de objetos de Jim Rumbaugh, Object Modeling Technique (OMT), Objectory Method de Ivar Jacobson, y el Método Booch de Grady Booch).

Es un proceso liviano, altamente iterativo, guiado por casos de uso, cuya clave es "Quitar la ambigüedad de los requerimientos y luego hacer un diseño claro" y que puede aplicarse a proyectos ágiles. Junto con Scrum pertenece al grupo de procesos diseñados para complementarse con otras metodologías como ser XP o RUP y para ayudar a eliminar ambigüedades en los requerimientos y diseñar antes de comenzar a codificar [ICONIX-PROC].

Estos métodos proveen procesos lógicos de bajo nivel, es decir, cómo codificar desde los casos de uso de una forma específica y repetible. No tiene especificaciones de alto nivel dado que considera que ésas son las áreas que más cambian de proyecto a proyecto y por ello, en las cuales se debe ser más flexible.

[29] http://www.martinbauer.com/articles/successful_web_development_methodologies

A continuación se mencionan los propósitos de cada artefacto en ICONIX, los casos de uso se emplean para definir el comportamiento de los requerimientos, el modelo de dominio para describir objetos reales y sus relaciones, el diagrama de robustez para quitar ambigüedad en los requerimientos y el diagrama de secuencia para asignar comportamiento.

Proceso

El proceso se puede dividir en siete pasos.

1) Identificar los objetos del mundo real conformando el modelado del dominio.

Este paso se puede dividir en dos etapas. En la primera se identifican los objetos del dominio real y sus relaciones que se representarán en un diagrama de clases de alto nivel. En la segunda, puede construirse un prototipo o recolectar información sobre los sistemas legacy que se quieren reconstruir. Al finalizar esto, ya se tendrá a todos los participantes del proyecto hablando el mismo vocabulario y casi todo el dominio del problema relevado, luego éste crecerá a medida que el proyecto avance.

2) Definir los requerimientos conformando casos de uso.

Los requerimientos de comportamiento son definidos mediante casos de uso. Al igual que el paso anterior, puede dividirse en etapas. En la primera se identifican los casos de uso usando diagramas de caso de uso, en la segunda se organizan en grupos, en la tercera se identifican los requerimientos funcionales a los casos de uso y objetos del dominio y por último en la cuarta, se escriben las descripciones de los casos de uso.

3) Efectuar un análisis de robustez para quitar la ambigüedad en casos de uso e identificar huecos en el modelo de dominio.

Se toma como entrada el texto de los casos de uso y se intenta hacer un primer modelado del sistema. Sirve para identificar objetos faltantes. Este paso se divide en una primera etapa en la que por cada caso de uso se identifican los objetos del escenario, se actualiza el dominio de negocio con dichos objetos y se actualiza el texto de los casos de uso para que se corresponda con el diagrama de robustez. La segunda etapa se basa en actualizar los diagramas de clase.

4) Asignar comportamiento a los objetos plasmándolo en un diagrama de secuencia.

Aquí es donde comienza el diseño. Generalmente se usa un diagrama de secuencias para su representación. Por cada caso de uso se creará un diagrama de secuencias.

Se divide en tres fases, en la primera se identifican los mensajes que se pasarán entre objetos y los métodos asociados. En la segunda, se dibuja el diagrama y en la última se actualizan los diagramas de clase con los atributos y operaciones encontrados.

5) Finalizar el modelo estático construyendo un diagrama de clases.

Se genera un diagrama estático con los diagramas de clase. Este diagrama contiene datos de implementación.

Se divide en dos etapas, en la primera se agrega información detallada de diseño (visibilidad y patrones por ejemplo) y en la segunda se verifica que se esté cumpliendo con todos los requerimientos.

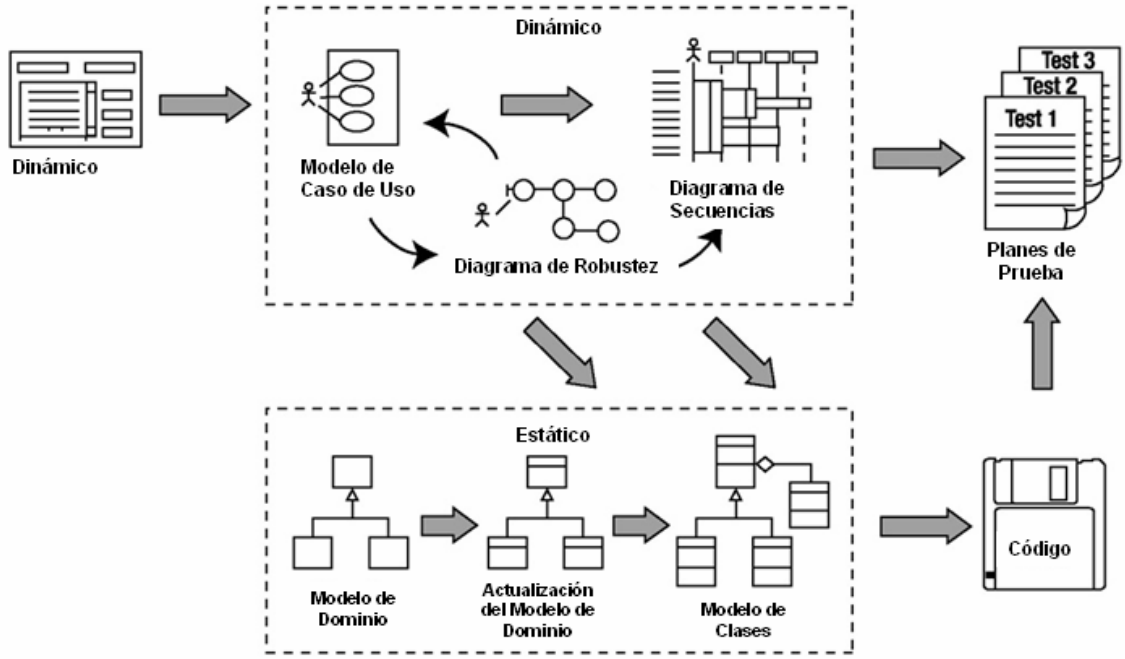
6) Codificar.

Aquí es donde se codifica el sistema, siendo ideal que los programadores hayan estado involucrados en tareas de diseño. Consiste en la generación del código y efectuar las pruebas unitarias.

7) Finalizar el sistema y hacer las pruebas de aceptación de usuario.

En este paso un grupo de Quality Assurance (QA) efectuará las pruebas del sistema de y aceptación de usuario.

A continuación se agrega una gráfica del proceso explicado anteriormente.



Proceso ICONIX. Fuente [ICONIX-PROC]

Los hitos que marcan la finalización de un paso y continuación al siguiente son:

1) Hito 1: Revisión de requerimientos: para asegurarse de que los casos de uso se correspondan con las expectativas de los clientes antes de comenzar con el diseño[30].

2) Hito 2: Revisión de diseño preliminar.

3) Hito 3: Revisión del diseño detallado y crítico.

4) Hito 4: Entrega

[30] http://iconixprocess.com/iconix-process/

Ventajas.

Su principal objetivo es reducir la ambigüedad desde los requerimientos y luego realizar el diseño más claro que se pueda lograr [ICONIX-PROC].

Posee una tendencia a alivianar los procesos mediante el pasaje rápido desde los casos de uso a la codificación con la menor cantidad de pasos posibles. Identifica qué documentos son temporales (aquellos que puedan, por ejemplo, ser abandonados una vez que se usaron) y qué documentos deben mantenerse actualizados [XP-REFACT].

Los refinamientos que se producen en cada iteración, tal como se ha mencionado en otras metodologías iterativas, provocan entre otras cosas, que los equipos de desarrollo ganen conocimiento y experiencia[31].

El uso de una herramienta de modelado y la construcción de los diagramas antes mencionados permite un mayor entendimiento del dominio del negocio mediante un lenguaje común (UML).

Limitaciones.

Requiere un gran diseño detallado up-front y revisión del diseño por desarrolladores de alto nivel de conocimiento antes de la codificación, para evitar el tiempo y recursos que podrían ser necesarios en un futuro refactor.

En ICONIX sin embargo, el equipo de desarrollo debe ser responsable de asegurarse que las excepciones no rompan el sistema basándose en los casos de uso antes planteados [XP-REFACT]. Esto presentaría una pequeña diferencia con XP. XP es dirigida por historias escritas por usuarios, que no podrían escribir un caso de uso y luego éstas son empleadas para construir los casos de uso. En ICONIX directamente se escriben casos de uso bien detallados, describiendo el comportamiento normal y el de las excepciones desde la perspectiva del usuario.

3.2.9 Comparaciones:

En base a lo visto a lo largo de la sección de metodologías de desarrollo ágiles y teniendo en cuenta los resultados de la encuesta [AGILE-SURV] se pueden obtener las siguientes conclusiones:

a) Las principales razones por las cuales se adoptan metodologías ágiles en el desarrollo de software son: El manejo de los cambios de prioridades, acelerar los tiempos de

[31] http://pdf.rincondelvago.com/modelamiento-de-datos_iconix.html

los negocios, incrementar la calidad de software, incrementar la productividad, alinear la tecnología y las necesidades de los negocios, reducir los riesgos dentro de un proyecto y por último mejorar la visibilidad en el proyecto.

b) Las mejoras alcanzadas con la utilización de metodologías ágiles en sus procesos de desarrollo: Incremento en la productividad, reducción de defectos en el software, aceleramiento de tiempos y reducción de costos.

c) Las mayores barreras de adopción de éstas metodologías son: la resistencia al cambio, personal con experiencia en éllas y límites de las organizaciones.

d) Una de las mayores limitaciones en la utilización de las metodologías ágiles se produce cuando el cliente no está dispuesto a participar activamente en el proyecto. Esto puede producirse debido a diferentes factores; por falta de tiempo, falta de impulso desde el nivel jerárquico, etc.

e) Generalmente la experiencia en la puesta en marcha de éste tipo de metodologías ronda entre los 2 y 5 años.

f) La metodología más utilizada actualmente es Scrum, luego Scrum con híbrido de XP, XP, otros híbridos y por último DSDM.

g) Si bien la mayoría de las personas que respondió la encuesta pertenecen a empresas con más de 250 empleados, la mayoría trabaja en equipos de no más de 10 miembros y en sus empresas poseen equipos de trabajo distribuidos.

h) De las metodologías más utilizadas y estudiadas en este trabajo se muestra a continuación un cuadro comparativo de las características principales de cada una de ellas.

Metodología	XP	Scrum	Crystal	FDD
Tamaño del equipo	2 a 10.[32]	1 a 7	Variable	Variable
Duración de cada Iteración	Menos de un día ó un día hasta un mes	4 semanas	Menos de 4 meses	Menos de 2 semanas
Soporte Distribuido	Si se emplean técnicas como video-conferencias en equipos distribuidos.	Adaptable	Si	Adaptable
Criticidad del Sistema	Adaptable	Adaptable	Todos los tipos	Adaptable

[32] Eventualmente se puede armar un equipo de "segundo nivel" que une a las "parejas" cliente-manager de cada equipo en una especie de "meta-equipo". De esta forma se pueden hacer sistemas complejos con base en subsistemas de mediana complejidad hechos con XP.

4. ARQUITECTURAS ORIENTADAS A SERVICIOS

4.1 Introducción

En este capítulo se define el concepto de SOA. Se detalla su surgimiento, elementos que la componen, sus ventajas y desventajas. Esto servirá de base para el capítulo siguiente donde se desarrollan las metodologías involucradas con SOA.

4.2 Antecedentes

Hacia la década de los 70 y 80, el sistema de información reinante era el mainframe, donde todos se conectaban a un computador central mediante terminales bobas. Entre mediados de los años 80 y principios de los 90 comenzó a popularizarse la arquitectura "cliente-servidor" que permitía usar las recién salidas computadoras personales, como integrantes de grandes sistemas de información, donde las aplicaciones corrían distribuidas, pero los datos estaban centralizados en un servidor de bases de datos.

Pensando en los años ´90, si un usuario deseaba ejecutar un programa, previamente debía instalarlo en su máquina. Con la evolución de las aplicaciones web, éstas pudieron ser accesibles por una gran cantidad de usuarios a través de múltiples sistemas operativos, siendo construidas en HTML. Así es como las aplicaciones se hicieron disponibles para todos. De esta forma, el acceso se tornó hacia fines de los ´90, en lo que conocemos como ubicuo, buscando que las nuevas tecnologías permitiesen una mejor entrega de aplicaciones y diversas formas de utilizarlas.

Las tecnologías EAI (Enterprise Application Integration – Integración de Aplicaciones Empresariales) empleadas para conectar la lógica de negocio sobre los sistemas y empresas, comenzaron a tornarse problemáticas. Hasta que a comienzos de los años 2000, la especificación XML denominada Simple Object Access Protocol (SOAP), simplificó dicha integración por medio de los servicios web.

Con la evolución de las tecnologías, los servicios web se hicieron más avanzados y los lineamientos de arquitectura recomendados para lograr el éxito, fueron creciendo en lo que denominamos SOA.

A continuación se agrega un pequeño resumen de la evolución técnica de SOA, partiendo desde el momento en que se desarrollaban aplicaciones monolíticas, cuando los primeros sistemas a gran escala fueron desplegados, y luego desde la evolución de lo

Procedural a la Orientación a Objetos, como los primeros pasos en la evolución SOA. Extraído de [SOA-ADOBE].

Paso evolutivo	Ejemplo
Monolítico.	Aplicaciones de gran escala, usando metodología de codificación procedural.
Estructurado u Orientado a Objetos (OO).	División de aplicaciones en unidades lógicas basadas en funcionalidad. Los primeros pasos de SOA.
Cliente – Servidor.	La progresión lógica de OO, empaquetando grupos de funciones en una computadora (servidor) e invocándolas luego desde otra (cliente).
3 Capas.	Se agrega una capa extra de interacción, creando una interfase agnóstica a un ambiente específico del servidor. Por ejemplo, si se hace una petición http, la capa del medio traslada la petición get() http al formato nativo necesario para obtener el recurso solicitado.
n Capas.	Llamadas request-response entre aplicaciones. El desarrollo de portales recae en este concepto. Por ejemplo J2EE[33].
Objetos Distribuidos.	Sistemas ubicuos y heterogéneos, de muchos objetos ortogonales y distribuidos, más que una simple interacción entre 2 o 3 computadoras. Por ejemplo CORBA[34].
Componentes.	Se agregan objetos en componentes lógicos que realizan funcionalidades específicas (a menudo mapeadas a componentes o constructores de requerimientos) e interfases a estos componentes. Un ejemplo es un servidor de base de datos usado como un componente de persistencia de datos para un software CRM.
Service Oriented Components.	Un ambiente ubicuo de componentes ortogonales que interactúan en un ambiente basado en pares, que a menudo emplean service provider proxies (interfases basadas en estándares aceptados) para ofrecer servicios.

Tal como hemos visto anteriormente, la Arquitectura Orientada a Servicios es una evolución de la Arquitectura Basada en Componentes, el Diseño basado en Interfase (Orientación a Objetos) y Sistemas Distribuidos de los '90, como por ejemplo: J2EE, CORBA y DCOM e Internet. Tanto .NET, J2EE, CORBA y ebXML[35] son implementaciones SOA. SOA no significa específicamente Web Service.

[33] http://es.wikipedia.org/wiki/J2EE
[34] http://www.corba.org/
[35] http://www.ebxml.org/

En la Arquitectura Orientada a Componentes las funcionalidades del todo son divididas en funciones más pequeñas, cada una encapsulada en un componente. Su principal ventaja es que permite que componentes específicos sean reutilizados facilitando esto, la manutención de los mismos. Además, el objetivo de las Arquitecturas Orientadas a Componentes en la ingeniería de software, es incrementar la productividad, calidad y mejorar el tiempo de desarrollo del software gracias al despliegue de componentes estándares y de la automatización de la producción. Define un framework que a su vez define requerimientos estructurales para la conexión y opciones de composición y requerimientos orientados al comportamiento para la colaboración de componentes. De esta forma, un modelo de componentes provee una infraestructura la cual implementa mecanismos de persistencia, intercambio de mensajes, seguridad y versionado. La idea es construir unidades de software intercambiables a través de interfases bien definidas [SOA-SOC].

Los Sistemas Distribuidos son una extensión de las Arquitecturas Basadas en Componentes y hace referencia a componentes que pueden existir en diferentes ubicaciones.

4.3 Concepto

Existen muchas definiciones acerca de SOA. Aun así, normalmente se suele confundir y tratar a Arquitectura Orientada a Servicios (SOA) como una tecnología de desarrollo de software. Sin embargo, SOA es un concepto de diseño de arquitectura que trata de alinear a las Tecnologías de Información, con el propio negocio de la organización.

“SOA es un paradigma arquitectural para componentes e interacciones de un sistema o patrones entre ellos. Un componente ofrece un servicio que espera en un estado de “listo”. Otro componente puede ser dicho servicio a través de la aceptación de un contrato de servicio [SOA-ADOBE]”.

La creación de servicios y funcionalidades de negocio deben ser fácilmente reutilizables, débilmente acoplados, altamente interoperables, seguros y lo más importante de todo, con una arquitectura basada en estándares. [DEF-SOA]

Bajo esta infraestructura es necesario que los servicios estén bien definidos, de forma que puedan publicarse y luego ser consumidos y combinados dinámicamente.

Otro de los aspectos fundamentales de SOA es enfocarse, en la mayor medida posible, en cumplir los requerimientos del negocio y usuarios, brindando soluciones alineadas bajo una visión global del proceso. Para cumplir con esto, SOA proporciona recomendaciones para lograr los objetivos de una organización en cuanto al desarrollo de aplicaciones. Sin embargo, para hacer uso de sus componentes no es necesario conocer todos los detalles.

4.4 Componentes.

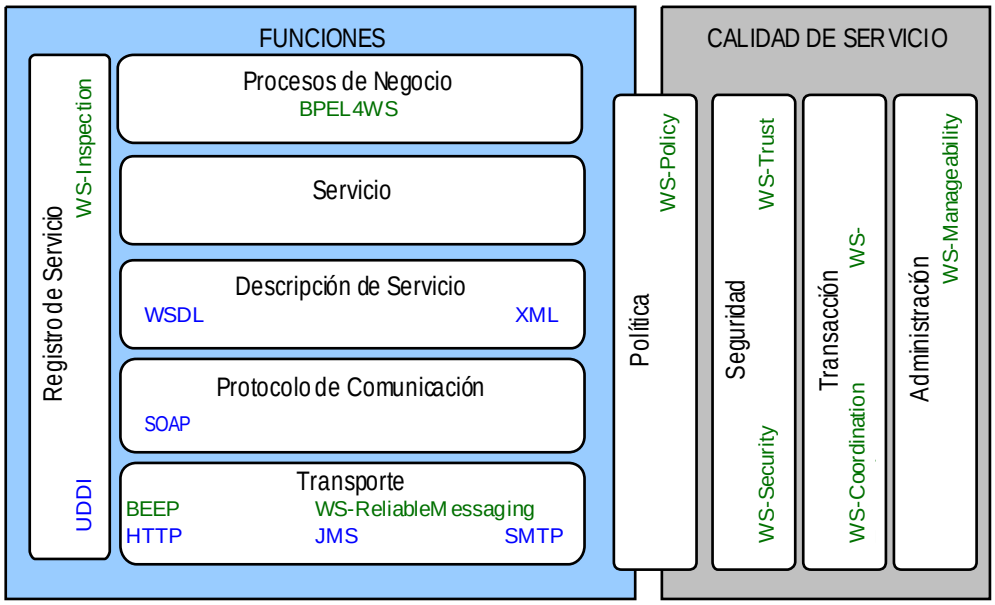
A continuación se incluirá una breve descripción de la arquitectura básica de SOA y los elementos que involucra.

Si bien la mayoría de las arquitecturas involucran los componentes que se mencionan a continuación (un proveedor de servicio, un consumidor de servicio y algún tipo de mensajería), no todos estos conceptos deben ser incluidos para formar una arquitectura SOA, sino que, teniendo en cuenta los conceptos mínimos de SOA se puede diseñar una aplicación que es compatible con SOA [SOA-ADOBE].

Según [SOA-BATUTA], los elementos de una arquitectura SOA se pueden clasificar en funcionales y de calidad de servicio.

En azul se indican aquellos que hoy día son estándares.

En verde se marcan aquellos estándares que están en todavía están en proceso de surgimiento [IBM-PATTERNS].



FUNCIONALES

Transporte

Es el mecanismo por el cual una petición de servicio es llevada desde una entidad consumidora a otra proveedora de dicho servicio, y a su vez, las respuestas desde los proveedores a los consumidores.

Protocolo de Comunicación

Es un mecanismo acordado a través del cual un proveedor de servicios y un consumidor comunican qué está siendo solicitado y qué está siendo respondido [SOA-BATUTA].

Descripción del Servicio

Cada servicio puede incluir una forma de definición de servicio.

Una descripción de servicio:

- Especifica la forma en que un consumidor de servicio interactuará con el proveedor del servicio, especificando el formato de consultas y respuestas desde el servicio.
- Indica lo que el servicio hace, la forma en que se vincula, si se debe emplear para con el un protocolo de seguridad y en ese caso, cuál.
- Permite la interacción y visibilidad entre participantes en interacción de servicios.
- Posibilita que potenciales participantes puedan construir sistemas que usen servicios y de esa forma puedan luego ofrecer servicios compatibles.
- Permite a los potenciales consumidores tener visibilidad de información crítica (si el servicio es alcanzable, si realiza determinadas funciones, si opera bajo determinadas políticas, etc.) de forma tal que le permita decidir cuándo emplear un servicio.

Las buenas prácticas sugieren que una descripción de servicio debería representarse empleando estándares. La ventaja de esto es que se pueden utilizar herramientas comunes de procesamiento como ser un motor de descubrimiento de servicios.

Los estándares son Web Service Description Language (WSDL)[36] de W3C que estando expresado en XML, describe cómo acceder al servicio, de qué funciones dispone, qué argumentos necesita y qué devuelve cada uno y luego Collaboration Protocol Profile de ebXML[37].

[36] http://www.w3.org/TR/wsdl
[37] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-cppa

Servicio

Antes de dar una definición de Servicio, se aclarará que el término *Servicio* no implica Servicio Web, sin embargo los servicios web, J2EE y .NET son implementaciones conocidas de Servicio.

Es un mecanismo que permite el acceso a una o más capacidades. Es provisto por una entidad llamada Service Provider a otros usuarios, sin necesidad de que éste conozca los potenciales consumidores del servicio y pueda demostrar el uso del servicio más allá del alcance concebido originalmente por el proveedor [OASIS-RM].

El acceso a un servicio se efectúa mediante una interfaz que especifica cómo es el acceso a las capacidades del servicio y respeta restricciones y políticas establecidas por el descriptor del mismo.

La invocación a un servicio puede resultar en el retorno de una respuesta con información solicitada y/o producir un cambio de estado en una entidad determinada.

Según Thomas Erl, los principios de la Orientación a Servicios son [ERL-DSTRA]:

- Los servicios deben ser reutilizables.
- Los servicios deben proporcionar un contrato formal.
- Los servicios deben tener bajo acoplamiento. Eso implica que [MICROSF-LC]:
  - Usando un recurso solo vía el servicio publicado y no ubicando directamente la implementación que hay detrás de él se conseguirá: que los cambios en la implementación del proveedor del servicio no afectarán al servicio consumidor.
  - El servicio consumidor podrá elegir una instancia alternativa del mismo tipo de servicio sin modificar su aplicación que hace la solicitud, aparte de la dirección de la nueva instancia.
  - El servicio consumidor y el proveedor no tienen que tener la misma tecnología para la implementación, interfase, o integración cuando se utiliza el servicio web si bien los dos están limitados a usar el mismo protocolo de servicio web.

Por otra parte, el nivel de acoplamiento puede aplicarse a dos capas. En la **capa tecnológica** se tiene en cuenta la integración con la plataforma y el nivel de red. Por ejemplo, si se conecta vía J2EE o .NET. Esto involucra computación distribuida o productos de mensajería. En la **capa de aplicación** se considera cómo se conectan las aplicaciones unas con otras y no siempre el uso de protocolos para servicios web permite esto (por ejemplo si se tienen en cuenta servicios web provistos por una aplicación empaquetada como SAP). Por ello, se requiere separar la capa de servicios del negocio de la tecnología y de la aplicación y diseñar SOA aparte de un requerimiento del negocio para formas específicas de adaptabilidad.

- Los servicios deben permitir la composición.
- Los servicios deben ser autónomos.
- Los servicios no deben tener estado.
- Los servicios deben poder ser descubiertos.

Proceso de Negocio

Es un conjunto de servicios que se invocan en una secuencia determinada para satisfacer un requerimiento de negocio.

En términos de SOA, consisten en una serie de operaciones las cuales son ejecutadas en una secuencia ordenada de acuerdo a determinadas reglas de negocio. El secuenciamiento, selección y ejecución de operaciones se denomina servicio o coreografía de procesos. Desde el punto de vista del modelado, es un desafío cómo las operaciones bien diseñadas, los servicios y la abstracción de procesos pueden ser caracterizados y construidos sistemáticamente [SOA-ELEM].

Registro de Servicios

Es un repositorio donde se alojan descripciones de servicios y datos que entidades de servicio pueden emplear para publicar los servicios que dispone y para que luego, entidades consumidoras puedan descubrir servicios disponibles.

CALIDAD DE SERVICIO:

Política

Es un conjunto de condiciones o reglas bajo las cuales un proveedor de servicio publica sus servicios para que estén a disposición de entidades consumidoras.

Seguridad

Es un conjunto de reglas que pueden aplicarse para la autenticación, autorización y control de acceso a consumidores.

Transacciones

Es el conjunto de atributos que podrían aplicarse a un grupo de servicios para entregar un resultado consistente.

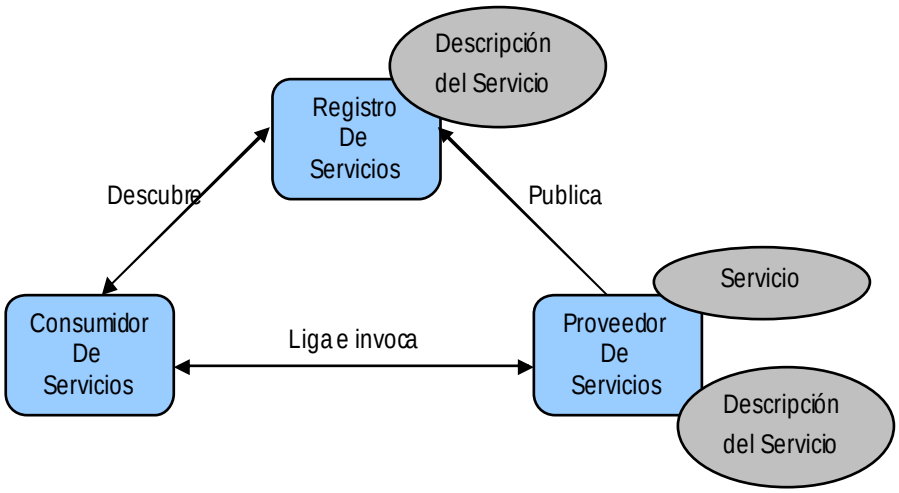
Administración

Es el conjunto de atributos que podrían aplicarse para manejar los servicios proporcionados.

Publicación y descubrimiento de Servicios

Básicamente consiste en que un consumidor (aplicación, módulo de software u otro servicio [SOA-BATUTA]) que requiere un servicio que cumpla con un determinado criterio, pueda localizarlo de forma dinámica mediante una consulta a un registro de servicios. Si el servicio existe (es decir, un servicio proveedor creo el servicio y publicó su interfaz e información de acceso en el registro de servicios), el registro proporciona al consumidor la interfaz de contrato y la dirección del servicio proveedor.

A continuación se muestran en un diagrama las entidades y roles que intervienen y se detallan las acciones de publicación y descubrimiento de servicios.



Publicación

La descripción de un servicio debe hacerse publica y de una forma accesible a potenciales consumidores. Una de las varias formas de hacer esto es por medio de la técnica *Pull and Push*.

Con el método *Pull*, los potenciales consumidores solicitan al proveedor del servicio el envío de la descripción del servicio. Luego, con el método *Push*, el proveedor del servicio envía la descripción del servicio a los posibles consumidores.

Existen otros modelos para la metodología *Push* que definen la forma en que el proveedor del servicio envía los descriptores de servicios a los potenciales consumidores. Por ejemplo:

Unicast: el mensaje sólo se envía a un solo destinatario.

Multicast: es un patrón de comunicación paralela en el cual la entidad origen envía el mensaje a un grupo de destinatarios.

Broadcast: se envía una transmisión a todos los posibles consumidores de mensajes.

"Si la intención de una infraestructura SOA es escalar más allá de algunos servicios, es deseable entonces que implemente específicamente un agente neutral, proveniente de un tercero, para la publicación y descubrimiento de servicios." [SOA-ADOBE]

Descubrimiento

Ocurre cuando un potencial consumidor toma conocimiento de la existencia de un servicio, sus términos y parámetros.

Registro/Repositorio

Es un componente donde los usuarios pueden almacenar y administrar artefactos que son requeridos y compartidos por varios usuarios para el funcionamiento de la empresa. Estos artefactos podrían ser esquemas XML, contratos de servicios, etc. El registro se entera de todas las modificaciones que se producen sobre cualquiera de los artefactos que almacena.

Registro de servicios

El **Registro de Servicios** (Service Registry en Inglés) es responsable de hacer disponibles servicios a consumidores, la interfaz de un servicio proveedor y de la implementación de acceso a la información. Por su parte, la implementación de un registro de servicios debe realizarse teniendo en cuenta el alcance. Por ejemplo existen registros de servicios disponibles en Internet para una audiencia sin restricciones o también aquellas que son del tipo privadas, donde solo pueden acceder usuarios dentro de una compañía por medio de una Intranet [IBM-PATTERNS].

Directorio de Servicios

Es una interfaz que provee información para vincular artefactos. Una entidad que sea propietaria o controle un artefacto podrá crear una entrada en éste directorio y crear una referencia a este artefacto y explicar la forma de enlazarlo. De esta forma, quedará disponible para que otra entidad consumidora, pueda acceder a dicha referencia y alcanzar ese artefacto. El inconveniente que posee un directorio es que no posee control sobre las modificaciones en los artefactos por lo tanto no puede comunicarlo a los usuarios.

A su vez, tanto el directorio de servicios como los registros / repositorios permiten que el contenido de una implementación sea replicado o referenciado desde dentro de otras implementaciones [SOA-ADOBE].

WSDL, UDDI y SOAP son piezas fundamentales en SOA. SOAP (siglas de Simple Object Access Protocol ó Protocolo de Acceso Simple a Objetos) como capa de transporte,

define un protocolo estándar XML para la interacción básica entre servicios, WSDL es un lenguaje común para describir servicios y UDDI provee la infraestructura necesaria para descubrir y publicar servicios. De ésta forma y mediante éstas especificaciones las aplicaciones pueden interactuar siguiendo un modelo débilmente acoplado e independiente de la plataforma. Si bien SOAP es el mecanismo por defecto para los servicios web, existen otras tecnologías para lograr el enlace de un servicio. Un consumidor puede alcanzar un servicio web en un registro UDDI, tomar su WSDL con la descripción del servicio y luego invocarlo usando SOAP. A continuación se da un breve detalle de cada una éstas tecnologías y su aporte en SOA.

SOAP

Ya hemos visto que una de las características más importantes que aporta SOA es el bajo acoplamiento entre los servicios. El bajo acoplamiento debe trasladarse también a nivel de la capa de protocolos, y para esto, la infraestructura de comunicación usada dentro de SOA debe ser diseñada de forma tal que sea independiente de la capa de protocolos subyacente. Cuando se diseña un framework de comunicación que soporte esto (la independencia de los protocolos) el reuso de código es uno de los mayores beneficios, además de que permite que, al hacer el reemplazo de un protocolo subyacente, sea más fácil el deployment de un nuevo protocolo sin necesidad de, por ejemplo, reiniciar los servers y causar frustración en el cliente[38].

A pesar de lo comentado anteriormente y tal como se venía adelantando, SOA podría ser implementado sin emplear servicios web, XML ni SOAP. Por ejemplo podría implementarse mediante HTTP REST[39] que está basado en HTTP y que requiere procesamiento XML el cual hoy día es soportado por la mayoría de los lenguajes de programación y plataformas[40]. REST fue introducido por Roy Fielding[41] al describir una arquitectura web basada en recursos. Un recurso es cualquier cosa localizable mediante una URI y que puede tener una o ninguna representación.

Un servicio web REST posee determinadas restricciones como por ejemplo[42]:

- Las interfases están limitadas a HTTP. Se definen las semánticas:
    - HTTP GET se usa para obtener la representación de un recurso mediante una URI.
    - HTTP DELETE se usa para eliminar las representaciones de un recurso.

[38] http://articles.techrepublic.com.com/5100-22-5198198.html
[39] http://www.xml.com/pub/a/ws/2002/02/20/rest.html
[40] http://www.devx.com/codemag/Article/28254
[41] http://es.wikipedia.org/wiki/Roy_Fielding
[42] http://webservices.xml.com/lpt/a/1292

- HTTP POST para actualizar o crear representaciones de recursos.
- HTTP PUT para crear una representación de un recurso.

- Generalmente se emplean mensajes XML.
- Los mensajes simples pueden ser codificados con una codificación URL.
- Los servicios y servicios proveedores deben ser un recurso. El consumidor puede ser un recurso.

A continuación se agregan detalles acerca del protocolo SOAP.

Fue creado en 1998 por Microsoft, IBM y otros. Actualmente es mantenido por la W3C[43] (World Wide Web Consortium) y XML Protocol Working Group[4445]. Puede usarse con otro protocolo que soporte la transmisión de datos en forma de XML desde un sistema emisor a otro receptor sin requerir el uso de HTTP, HTTPS o una conversación del tipo request/response desde la versión 1.2. Por esta razón, los procesos SOAP son agnósticos e independientes en cuanto a protocolo. Por ejemplo, IBM y Microsoft implementaron mensajes SOAP sobre SMTP [SOAP-NET].

Permite el intercambio estructurado de mensajes basados en XML entre un servicio proveedor, un servicio consumidor y un registro de servicios. Fue creado de forma tal que sea independiente de cualquier modelo de programación o semántica específica de alguna determinada implementación [SOA-PATTERNS].

SOAP soporta el intercambio de mensajes vía RPC (Remote Procedure Call) y está diseñado de forma tal que permite el bajo acoplamiento mediante la independencia de protocolo, de lenguaje, plataforma y sistema operativo[46].

Los mensajes SOAP contienen información para controlar el movimiento del mensaje en su recorrido. Esta información es independiente de protocolo de transporte que se use para transmitir el mensaje SOAP.

Consiste de tres partes:

- El **formato de mensaje** SOAP es una envoltura que contiene cero o más encabezados y exactamente un cuerpo. El *envoltorio* es el elemento más exterior (el que contiene a los demás) de un documento XML, y permite contener información de control, la ubicación del mensaje y el mensaje en si. El encabezado es opcional y contiene información de control como ser atributos referentes a la calidad del servicio. El cuerpo es obligatorio (mandatorio) y contiene la identificación del mensaje y sus parámetros. El identificador

[43] http://en.wikipedia.org/wiki/World_Wide_Web_Consortium
[44] http://www.w3.org/2000/xp/Group/
[45] http://es.wikipedia.org/wiki/SOAP
[46] http://articles.techrepublic.com.com/5100-22-5198198.html

namespace para los elementos y atributos en esta sección es "http://schemas.xmlsoap.org/soap/envelope/"[47].

- **Reglas de Encabezado**: se usan para expresar instancias de tipos de datos de aplicaciones definidas. SOAP define un lenguaje de programación independiente del tipo de dato basado en esquemas xsd (XML Schema Descriptor), más reglas de codificación para todos los tipos de datos definidos en este modelo.
- **Representación RPC** que es la convención para la representación de llamadas a procedimientos remotos (RPC) y respuestas.

WSDL

Debido a que la interacción entre el consumidor y proveedor de un servicio debe ser diseñada completamente independiente de la plataforma y del lenguaje, es necesario un documento (usualmente Web Service Description Language) que defina y describa la interfaz del servicio sobre un protocolo de red (HTTP normalmente) [IBM-PATTERNS].

El uso de contratos y su posterior publicación en un repositorio central, es el punto más común para comenzar la estandarización en SOA. En la descripción de interfaz, ya sea CORBA IDL (Interfase Definition Language), WSDL ó un formato XML a medida, el contrato debe contener toda la información relevante para el uso del servicio en las aplicaciones, por ejemplo: precondiciones, casos especiales, errores potenciales, información de versionado, y permisos de acceso [SOA-BPPH].

WSDL surgió en septiembre del año 2000 de la mano de las empresas: Microsoft, IBM y Ariba. Se basa en los lenguajes de definición NASSL44 (Network Accesible Services Specification), de IBM, y SCL45 (SOAP Contract Language) de Microsoft. En Marzo de 2001, estas compañías, con el apoyo de algunas otras, enviaron la versión WSDL 1.146 al W3C, donde fue publicado como una nota por lo que formalmente no es un estándar del W3C. Básicamente define qué funciones se pueden invocar, los tipos de datos que éstas funciones utilizan, el protocolo de transporte para el envío recepción de mensajes y cómo acceder a los servicios (URL para acceder a los servicios) [SOA-BATUTA].

Por otra parte, y para finalizar, se incluyen los elementos que compone un documento WSDL está compuesto y un breve detalle de cada uno según [EAI-LINTH2]:

- Definiciones de tipos, para elementos de datos (generalmente usando esquemas XML).
- Definiciones de mensaje incluyendo uno o más elementos de datos tipados.
- Definiciones de operaciones que son descripciones abstractas de acciones soportadas por el servicio y define cuáles son los mensajes de entrada y salida.

[47] http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383494

- Definiciones de tipo de puertos que son una lista de operaciones soportadas por el servicio web.
- Definiciones de enlaces (bindings), que describen el vínculo entre tipos de puerto (portTypes) y protocolos (SOAP, HTTP GET/POST, etc.).
- Definiciones de servicio que lista un conjunto de enlaces.

Dentro de WSDL los servicios son una colección de endpoints (puntos de acceso a la red ya sea un proveedor o un consumidor de servicio ó puertos) dentro de una red y su definición abstracta. Este mecanismo favorece el reuso de definiciones abstractas o mensajes. Los mensajes son descripciones abstractas de información que fluye de una aplicación a otra y están separados de los enlaces (binding) de formato de datos. Los tipos de puertos son conexiones abstractas de una operación, que a su vez es una descripción abstracta de una acción soportada por un servicio. Los bindings son protocolos concretos y especificaciones de formatos de datos para una instancia de portType.

UDDI

UDDI (siglas de Universal Description, Discovery, and Integration ó Descripción Universal, Descubrimiento e Integración) es un directorio de servicios web donde se pueden publicar servicios, describir el tipo de servicio y realizar búsquedas [SOA-BATUTA].

Es una API del lado del cliente y una implementación servidora basada en SOAP que puede usarse para almacenar y tomar información de proveedores de servicios y servicios web [SOA-PATTERNS].

El registro UDDI juega un papel importante en una infraestructura de servicios web, aportando valor al negocio. Según OASIS[48] en una publicación denominada “Making the business case[49]”, UDDI ayuda a asegurar que las necesidades funcionales y prácticas de los desarrolladores, los requerimientos operacionales y de escalabilidad de los arquitectos de una organización, y las políticas de los negocios subyacentes no se contradigan; mas bien alinea a todas esas necesidades incrementando la flexibilidad, el reuso y el control. El reuso de código y las mejoras en el manejo de la infraestructura se implementa de forma estandarizada mediante:

- La publicación de información sobre los servicios.
- El descubrimiento de servicios,
- Determinación de la invocación de un servicio, seguridad, protocolos de transporte y parámetros y

[48] http://www.oasis-open.org/
[49] http://uddi.xml.org/business-case

- Proveyendo los medios para aislar a las aplicaciones de fallas o cambios en servicios invocados.

En esa publicación también se menciona el rol de UDDI en el desarrollo de servicios web y en una arquitectura orientada a servicios. A continuación se incluyen los detalles de cada una.

- El **rol que desempeñan los registros UDDI en el desarrollo de servicios web** está relacionado con las situaciones que enfrentan los desarrolladores cuando construyen servicios web (reuso, mantenimiento, etc.). Los registros UDDI ayudan a responder cuestiones del tipo: ¿Cómo pueden, los líderes de desarrollo, organizar y administrar sistemáticamente los servicios web sobre múltiples sistemas y equipos de desarrollo?, ¿Cómo pueden los desarrolladores manejar el movimiento de servicios a través de las fases de desarrollo (de codificación o pruebas a despliegue)?, y ¿Cómo pueden los programadores documentar las especificaciones de las interfases, transporte de mensajes y mecanismos de autenticación con otros grupos de desarrolladores? ¿Cómo pueden las aplicaciones externas absorber el impacto de los cambios producidos en otros servicios?. Los registros UDDI proveen una solución interoperable, basada en estándares para la documentación y publicación de servicios independientemente de la plataforma o ambiente. Ayuda a los desarrolladores a encontrar servicios compartidos y usar esos servicios dentro de sus propias aplicaciones.
- En **una arquitectura orientada a servicios** los arquitectos de software deben considerar: ¿Cómo pueden las aplicaciones críticas ser aisladas de cambios o fallas en servicios compartidos back-end? ¿Cómo puede una organización compartir información sobre servicios de una forma controlada que refleje sus propias reglas de negocio y políticas?. Los registros basados en UDDI proveen a los administradores IT una capa formal de indirección necesaria para el desarrollo y administración de aplicaciones orientadas a servicios. Provee una especie de firewall entre los servicios y las aplicaciones que los invocan, de forma que los administradores de sistemas pueden fácilmente manejar los cambios en el ciclo de vida de determinados componentes.

Por otra parte, los registros UDDI facilitan las necesidades operacionales y de governance; la versión actual de UDDI agrega soporte para funcionalidades como la autenticación y publicación / descubrimiento para registros pares.

Para finalizar, Joe McKendrick en su nota "UDDI set to emerge from the shadows of obscurity (UDDI necesita emerger desde las sombras de la oscuridad)"[50] indica las razones por las cuales la implementación de UDDI tardó su tiempo, basándose en una entrevista[51] realizada a Burton Group's. Anne Thomas Manes, la persona entrevistada, indica que no se

[50] http://blogs.zdnet.com/service-oriented/?p=812
[51] http://searchwebservices.techtarget.com/qna/0,289202,sid26_gci1242398,00.html

necesita tener UDDI para comenzar con servicios web ni tampoco para permitir la integración de aplicaciones, pero si se quiere hacer SOA se tiene que comenzar administrando el ambiente y UDDI permite esto mediante la comunicación sobre múltiples ambientes.

El uso de UDDI se ve en mayor medida hoy día debido a que las empresas se encuentran en la fase de puesta en producción, que es cuándo más se requiere un registro de servicios. Por otra parte, a medida que las empresas comienzan a desarrollar más y más servicios se les hace imperante contar con una herramienta que les permita administrarlos. Esto mismo pudo verse en el resultado arrojado en una encuentra de webServices.org[52] donde sobre un total de 1000 compañías, el 17% consideró el uso de UDDI en las primeras etapas de desarrollo y un 44% en las etapas avanzadas de deployment SOA (al menos 20 servicios compartidos entre dos o más unidades de negocio).

Algunas consideraciones:

Hay que tener en cuenta aquí que "SOA no significa SOAP" [SOA-SOC], dado que es posible desarrollar con otras técnicas que no sean SOAP. Si hay que proveer servicio a terceras partes la mejor elección sería mediante un web service. La orquestación de servicios no necesariamente requiere de web services, también se puede tener en cuenta otras tecnologías como ser JMS, EJB o CORBA.

Por otra parte, [SOA-SOC] también indica algunas desventajas sobre el uso de XML/SOAP. Con respecto al uso de XML indica que es bueno debido a que es independiente de la plataforma pero requiere la transferencia de datos que en gran volumen podrían incrementar el tráfico en la red y el parsing de los mensajes XML toma más tiempo que la serialización-deserialización de los datos enviados en formato binario sobre la red. Por otra parte con respecto a SOAP indica que su performance podría verse afectada teniendo en cuenta lo siguiente:

- Extraer la envoltura de SOAP del paquete tiene un costo de tiempo.
- Se tiene que usar un parser XML para identificar la información contenida en la envoltura SOAP.
- No es posible grandes optimizaciones con los datos XML.
- Las reglas de codificación SOAP hacen obligatorio incluir información SOAP en todos los mensajes enviados y recibidos.
- Codificando datos binarios en una forma aceptable a XML resulta en sobre carga de bytes adicionales como resultado de la codificación tanto como sobrecarga procesamiento de codificación-decodificación que afecta el rendimiento.

[52] http://www.webservices.org/

4.5 Roles.

Debido a la falta de madurez de las metodologías SOA (que se estudiarán en secciones posteriores) los roles aplicables a este tipo de arquitecturas podrían ser los mismos que los sugeridos por las metodologías de desarrollo planteadas anteriormente, como por ejemplo Scrum, XP, RUP, etc.

A continuación se indican los roles básicos definidos según [SOA-UY]:

- El **Arquitecto**, quién tiene responsabilidad en el área de Diseño pero participa también en el relevamiento de requerimientos y como coordinador del desarrollo;
- El **Analista**, participa en el relevamiento de requerimientos;
- El **Especialista Técnico**, tiene a su cargo el estudio e investigación de las tecnologías disponibles a utilizar en el proyecto.
- El **Implementador**, tiene a su cargo la codificación de la aplicación.

4.6 SOA y otras arquitecturas.

Podemos comparar SOA con otras arquitecturas.

SOA no contradice otras arquitecturas existentes. De hecho puede verse como una vista “macro” de Orientación a Objetos (OO) debido a que los dos modelos están basados en los mismos principios claves. OO introdujo el concepto de encapsulamiento permitiendo ocultar detalles de implementación de un objeto detrás de una interfase bien definida. Sin embargo, en OO, lo que un objeto hace está intrínsecamente atado a los datos en si mismo. Con el tiempo, los desarrolladores profesionales reconocieron las limitaciones de ésta aproximación y divisaron una alternativa en la cual el comportamiento pudiese ser duplicado y evolucionado más allá de los tiempos [QUOVADX].

SOA, a diferencia de otras soluciones de integración como EAI no se limita al uso de una herramienta o "plataforma de herramientas" para integrar aplicaciones, sino que sugiere una arquitectura ágil, escalable y completamente distribuida por toda la organización. En las arquitecturas SOA entre otras muchas funcionalidades, se integran aplicaciones al igual que hacen los EAIs, pero no se reduce a la integración de éstas dentro de una localización concreta, sino que va mas allá, va a los procesos de las organizaciones, a la gobernabilidad, al uso de tecnología estándar, a la integración en entornos distribuidos. [DEF-SOA].

ESB (Enterprise Service Bus), a diferencia de SOA es una tecnología o producto software. Puede definirse un ESB como la infraestructura que sirve como la columna vertebral de las Arquitecturas Orientadas a Servicios (SOA). Un ESB permite a una empresa, conectar, mediar, y controlar la interacción entre diversas aplicaciones y servicios a lo largo de entornos altamente distribuidos y heterogéneos.

SOA de por sí puede aportar una gran potencia y flexibilidad a una organización sin necesidad de utilizar un ESB, pero sin un ESB es muy complicada la gestión y mantenimiento de los procesos SOA y el escalado de la arquitectura en caso de ser necesario. [DEF-SOA].

En secciones anteriores también se ha visto que SOA es la evolución de las Arquitecturas Orientadas a Componentes. Si bien en algunas situaciones pareciera no divisarse claramente la división entre una SOA y una SOC se mencionarán las diferencias básicas entre ellas.

- Los servicios tienen que ser accesibles públicamente. Como se ha visto, la forma de hacer que sea posible descubrir y publicar servicios es a través de UDDI.
- Los servicios tiene que ser independientes de atributos específicos de la implementación, esto significa que al usuario no debería importarle si la versión del servicio es Java, .NET o Perl dado que la comunicación se basa en XML y probablemente con un protocolo SOAP.

4.7 Ventajas.

A continuación se mencionarán en primer lugar las ventajas que aporta una Arquitectura Orientada a Servicios en una organización y por último las desventajas.

La principal ventaja del modelo SOA es que facilita la integración de aplicaciones, aumenta la facilidad de adaptación, además de la posibilidad de disminuir costos de desarrollo al incrementar la reutilización.

Agilidad en los negocios:

Un diseño sólido de aplicaciones SOA hace considerablemente más fácil y eficiente incrementar, modificar y extender funcionalidades como las necesidades que surjan lo necesiten. Eso es debido a que las aplicaciones son diseñadas, desde un principio, con orientación al cambio y de ésta forma, el equipo de desarrollo podrá impactar los cambios en

pocos días o semanas, en comparación con los altos tiempos que insumen las aplicaciones tradicionales. [QUOVADX]

Interoperabilidad entre sistemas empresariales.

Sería imposible rescribir todos los sistemas, con el fin de lograr su comunicación, SOA propone una arquitectura que brinda esta capacidad y la de proveer soluciones a un ambiente de cambio total como el de los negocios.

Si una empresa utiliza paquetes de software de proveedores externos, se tornará sencillo adaptar nuevos procesos de negocio que incorporen nuevas funcionalidades y hagan uso de éstas aplicaciones mediante la publicación de interfaces a dichos paquetes.

Flexibilidad en Despliegues (Deployment Flexibility en inglés).

Con un modelo SOA, el dominio del negocio está separado completamente de su despliegue. Es decir, las decisiones de deploy y del negocio no están relacionadas, por lo tanto los desarrolladores pueden codificar y hacer pruebas en sus máquinas sin tener que luego, efectuar cambios en el código en el pasaje de ambiente.

Por otra parte, debido a que las aplicaciones SOA son modulares, no es necesario tener los componentes finalizados para realizar un deploy sino que esto puede hacerse de a pasos. Es decir, en una fase se despliega una parte, en la siguiente la otra, y así sucesivamente. De esta forma, las compañías tendrán rápidamente procesos disponibles de forma on-line. Esto mismo contradice la máxima de Marshal McLuhan y Quentin Fiore "*el medio es el mensaje*" por la cual se considera que el medio de comunicación es modelador de la conducta del ser humano, es decir el nuevo medio crea una nueva dimensión de interpretación de los mensajes. En contraposición, SOA es un nuevo medio que permite convertir mensajes "parciales" en una nueva interpretación del mensaje.

De acuerdo a "el medio es el mensaje", antes de existir los medios audiovisuales remotos, el ser humano tenía conocimiento sólo de lo que ocurría en su entorno cercano; la invención de la imprenta creó el pensamiento mecánico y separado de la acción, pero con la llegada del medio eléctrico, esta postura cambió. Los medios audiovisuales remotos rompieron las barreras comunicacionales de tiempo y espacio. Lo que antes se llamaba público (entes aislados, con puntos de vista diferentes), el medio eléctrico lo constituyó como entes relacionados entre sí, obligados al compromiso y a la participación[53]. Por lo tanto, los medios audiovisuales remotos (que pueden tener una base de funcionamiento eléctrica, óptica, química, biológica, etc.) adquieren importancia por su funcionalidad y no su

[53] http://apuntes.rincondelvago.com/marshall-mcluhan_procesos-comunicacionales.html

comportamiento interior. Por ejemplo los tambores de las tribus de la antigüedad eran, a su manera, medios audiovisuales remotos, una especie de radios de bajo ancho de banda.

Reuso.

Este es uno de los beneficios más importantes de SOA. El multiuso de servicios de integración es uno de los valores iniciales que puede brindar SOA a una organización.

Siempre ha existido el dilema de cómo hacer para reutilizar de la mejor manera posible el código. En SOA el principal problema aquí radica en la disponibilidad de código ya construido y en la reutilización en otros sistemas que tienen necesidad de funcionalidades similares. SOA permite que mediante un esquema de registro y publicación, las funcionalidades sean agrupadas y publicadas en servicios y luego, independientemente de la plataforma y la forma en que fueron construidos, puedan ser consumidos por otros servicios.

Por otra parte, el modelo SOA enfatiza el diseño sólido de aplicaciones, y pone especial foco en la fase de diseño. Esto significa que el equipo de desarrollo deberá tomarse el tiempo necesario para considerar los componentes que se reutilizarán. Pero una vez diseñados, serán construidos una sola vez y luego podrán ser reutilizados por otras soluciones a través de la empresa. De esta forma, se logra la reducción de la cantidad de código producido cuando se construye una nueva solución o se modifican soluciones ya construidas.

La forma de reutilizar esto es mediante la exposición de servicios siendo esto la clave de una arquitectura flexible. Una vez expuesto un servicio, otro puede consumirlo.

Si bien no es necesario conocer su ubicación física, es necesario que posean interfaces que gobernadas por contratos, definirán claramente el conjunto de mensajes soportados, su contenido y las políticas aplicables.

Cada servicio evolucionará de forma independiente con cada modificación o funcionalidad agregada, y esto también dará lugar a que evolucione el sistema en su totalidad.

El uso de servicios permite a los desarrolladores tomar ventaja sobre las aplicaciones legacy, las cuales pueden albergar los datos y funcionalidad de sistema, pero no pueden ser usados (debido a la complejidad que esto acarrearía) en la construcción de nuevas aplicaciones a menos que sean expuestos como servicios. La habilidad de re-usar la lógica central y la funcionalidad entre aplicaciones legacy mantiene un buen trato de tiempo y costos en la construcción de nuevas aplicaciones que trabajen cooperativamente con los sistemas legacy. [QUOVADX]

El reuso de módulos de sistemas legacy como proveedores de servicios, invalida completamente la "ortogonalidad" de SOA debido a que éstos sistemas no son ni

ortogonales ni heterogéneos (cada uno maneja diferentes estándares, lenguajes y operatorias). Aún así SOA sigue siendo útil dado que por su flexibilidad y adaptabilidad permite aprovechar las funcionalidades existentes que éstos sistemas existentes proveen.

Seguridad

Facilita la adopción de medidas de seguridad mediante la implementación y reuso de sistemas que permiten la autenticación, autorización y el encriptado de mensajes.

Puede emplearse SAML que consiste en un acuerdo de negociación formal entre dos partes. Contiene el entendimiento común sobre el servicio, disponibilidad, servicialidad, rendimiento, prioridades, responsabilidades, garantías, etc con el propósito de acordar un nivel de servicio[54].

Transparencia de la información.

Por otra parte, SOA puede ser un instrumento valioso para permitir la transparencia de la información partiendo desde los procesos de negocio y llegando hasta los servicios que implementan dichos procesos. Los organismos gubernamentales deberían estar obligados a ser gestionados en base a este tipo de transparencia y SOA es una buena base para permitir auditorias adecuadas. [IMAZ].

Los modelos de servicio como herramientas esenciales para el "refactoring" [SOAGILITY]

En SOA las aplicaciones de servicio deben ser vistos como componentes reusables, lo cual indica que se deberá contar con un desarrollo ágil debido al incremento en la cantidad de consumidores para dichas aplicaciones. A su vez, con el aumento de consumidores se incrementarán los cambios requeridos en las aplicaciones para que éstas puedan soportar las peticiones de todos los consumidores.

El desarrollo ágil consiste en el Modelo de Servicio, el modelo de los servicios, sus dependencias, coreografía y flujos. El modelo de servicio insume tiempo y definición de interfases. El modelo de servicio de una compañía podría mostrar debilidades debido a que sus responsabilidades no están definidas de la forma correcta implicando entonces el pasaje de responsabilidades de un servicio a otro y luego cambiar las interfases de éstos. Tanto en SOA como en el desarrollo ágil se alienta la reconstrucción (refactoring en Inglés).

En SOA el refactoring de servicios significa el cambio de interfases, cambiando las responsabilidades de un servicio a otro. Esto puede ser controlado mediante el modelo de

[54] http://en.wikipedia.org/wiki/Service_Level_Agreement

servicios herramienta esencial en el desarrollo ágil en SOA. Sin embargo hay que tener en presente que lo último que se debe pensar es en cambiar la interfaz del servicio por el impacto que esto conllevaría hacia todos los consumidores del servicio. Más sencillo sería modificar lo que se encuentra dentro del servicio como si fuese una caja negra.

4.8 Desventajas

No se tiene aún la perspectiva necesaria para determinar las desventajas, pero se puede anticipar el hecho de que, en realidad, una arquitectura SOA mal desplegada puede determinar que la empresa sea menos ágil y se ajuste menos a las exigencias de adaptabilidad que mencionamos más arriba [IMAZ].

Overhead

Debido a que las soluciones SOA generalmente acarrean gran overhead es necesario un análisis más profundo de los sistemas que involucran gran volumen transacciones debido a que el rendimiento del sistema podría verse afectado por la latencia de las solicitudes a cada servicio.

Contratos

Debido al estado de inmadurez de determinados aspectos de SOA, pueden encontrarse algunas limitaciones en lo que respecta al manejo de contratos. Actualmente WSDL no soporta todos los requerimientos funcionales y no funcionales de una organización. Las especificaciones WS-Policy proveen un contenedor interoperable para el intercambio de información sobre políticas, pero no provee soporte para especificar el comportamiento de las mismas[55].

Versionado

Al igual que en cualquier otro ciclo de vida del desarrollo, el concepto de versionado en SOA también es necesario dado que si bien no se espera que las interfases de los servicios cambien a menudo, es posible que esto suceda en ocasiones donde, por ejemplo, los sistemas backend cambien y luego los proveedores deban encontrar y establecer alguna forma de compatibilidad. El versionado implica que a medida que una nueva versión del servicio esté disponible, los clientes no tendrán que cambiar de versión a menos que necesiten las funcionalidades allí incluidas. Las implementaciones SOA deben proveer una forma mediante la cual permita a las versiones de los clientes coexistir con las nuevas

[55] http://dev2dev.bea.com/pub/a/2006/11/soa-service-lifecycle-run.html?page=last

versiones de los servicios. En caso contrario, todos los clientes deberán cambiar cada vez que se produzca un cambio en los servicios que se proveen.

El versionado en SOA tiene limitaciones debido a que no existe un sistema o producto que provea compatibilidad por siempre dado que no es posible financieramente o no es sustentable por mucho tiempo. Entonces, cualquier solución de versionado podrá eventualmente forzar a los clientes a migrar de una versión a otra, o saltar varias versiones para llegar a la última, cuando la versión en la que estaba se torne obsoleta. Documentar estas limitaciones sobre el versionado es importante y debe ser comunicado a los clientes mediante SLA (Service Level Agreement[56]) ó algún otro medio de comunicación de forma que los clientes puedan saber cuando deberán migrar su código a una nueva versión[57].

Metodologías

Tal como se menciona en el estudio [SOA-SURV] en general, las metodologías de desarrollo que han surgido hasta el momento son lo suficientemente recientes como para ofrecer un nivel de madurez aceptable. Por lo tanto es difícil de determinar cuál es la más apropiada, si es que existe, y cuál es la que aporta más valor a SOA.

Otra de las desventajas producto de la inmadurez del estado de SOA, es que se encuentran muchos documentos que hablan sobre metodologías SOA pero generalmente constan de publicaciones de empresas que intentan vender sus metodologías u ofrecerlas como estándares.

Diseño

Si no se efectúa un buen diseño puede darse el caso que determinados servicios evolucionen y se transformen en servicios críticos, y si éstos no están bien diseñados, cualquier cambio en ellos o en su interfaz puede provocar un alto impacto en el resto de los sistemas con los cuáles se vincula.

Costos

Por otra parte, una de las cuestiones a tener en cuenta son los **costos** involucrados en SOA. Libor[58] considera que el desarrollo de sistemas basados en SOA requiere altos costos en comparación con el desarrollo de una aplicación del tipo cliente/servidor o DCOM y menciona los gastos relacionados. Por otra parte considera que estos puntos se contrarrestarían en aplicaciones de gran escala. Costos:

[56] http://en.wikipedia.org/wiki/Service_Level_Agreement
[57] http://blogs.ittoolbox.com/emergingtech/memca/archives/soa-versioning-10876
[58] http://lsblog.wordpress.com/tag/soa/

- Una estructura compleja (bajo acoplamiento que requiere transformaciones de datos de forma asidua y las comunicaciones asincrónicas como medio de comunicación).
- Monitoreo y administración complejo como ser en procesos distribuidos, monitoreo de rendimiento, etc.
- El uso de comunicaciones asincrónicas para los desarrolladores puede ser más complejo de dominar y desde el punto de vista de las pruebas unitarias tiene un requerimiento de hardware más alto.

En la sección 4.11 se analiza en mayor detalle la relación costo-beneficio.

Reuso

El reuso puede estar limitado de acuerdo a las reglas del negocio y su impacto en el desarrollo de los servicios. En este sentido Libor[59] indica que si se siguen las sugerencias de diseño recomendadas y se agregan la menor cantidad de lógica de negocio en una capa de datos, luego se podrá reutilizar más veces este servicio. Por el contrario, a mayor cantidad de reglas el servicio será más especializado y como resultado tendrá menor reusabilidad.

4.9 Fallas en SOA

Según Joe McKendrick en su foro[60], no es tan trivial saber si un proyecto SOA está fracasando o no. Un indicio de falla podría ser cuando no se da lo proyectado en el ROI (retorno de inversión), pero existen ciertos cuestionamientos que también hacen pensar en las fallas de un proyecto SOA, por ejemplo:

- Pensar que porque utilizan web services se tiene SOA.
- Si sucede como cita Jim Kobelius que en una organización optan las prácticas SOA y finalmente se logra complejizar el ambiente, se incrementan los costos, si no se consiguen logros en solucionar incompatibilidades sobre diferentes plataformas o si la organización termina "comprometida" con un proveedor de soluciones determinado.
- Si se cae en los siguientes puntos de falla:
    - Ausencia de SOA "verdadera": cuando una organización tiene servicios y espera que SOA los envuelva en un mismo punto y termina siendo una arquitectura construida sin haber consultado primero a la gente que la usa, ó es tan segura y compleja que no alienta a que la gente confortablemente la use.
    - Falta del reuso de servicios o no compartir unidades de negocio ó si no se lleva un registro de esto.

[59] http://lsblog.wordpress.com/tag/soa/
[60] http://blogs.zdnet.com/service-oriented

- Si se gasta más dinero que el que se gana o no se lleva un registro adecuado de las ganancias.
- Si se tienen más bloqueos desde vendedores sabiendo que el propósito de SOA es independizarse de ellos. Una forma de darse cuenta si no se está haciendo lo que las buenas prácticas de SOA proponen, es si se tiene que esperar a un vendedor para mejorar una parte específica de una aplicación, ó si se está obligado a hacer una actualización.

4.10 Malas prácticas en SOA

En el trabajo publicado por [SOA-WPRAC] se incluyen una serie de malas prácticas que llevarían al fracaso de un proyecto con arquitectura SOA. A continuación se citan y describen brevemente cada una de ellas.

- Efectuar cambios sobre un estándar, en este caso SOAP, introduciendo una forma propietaria de mensajería podría traer los siguientes problemas además de impedir el reuso y la flexibilidad aportada por un estándar abierto:
  - Inhibición de la interoperabilidad limitando el alcance de SOA e incremento de los costos.
  - Personalización de las aplicaciones y servidores involucrados con esta nueva mensajería.
  - Pérdida de las posibles optimizaciones que brindan los estándares SOAP.
- Hacer "wrap" de servicios sin tener una estrategia. Debe tenerse cuidado al momento de usar un web service como "wrapper" de un servicio preexistente, dado que puede caerse en el caso de alto acoplamiento donde los consumidores deban conocer acerca de la implementación del mismo antes de invocarlos. El incorporar web services debe ser parte de una estrategia SOA de nivel organizacional sino se puede caer en problemas de rendimiento y / o de seguridad. La recomendación aquí es adoptar SOA de una forma incremental, primero incorporando los servicios que ya están funcionando y luego el resto bajo una política y estrategia indicada.
- Confundir la idea de SOA. Para evitar esto, tener en cuenta las siguientes tres cosas.
  - Antes de comenzar a adoptar SOA deben tenerse las razones correctas basadas en el negocio o en los objetivos IT.
  - Divisar la estrategia de adopción SOA junto con la planificación acerca de la forma en que se efectuarán las comunicaciones con proveedores y consumidores y demás impactos.

- Ser conscientes de que la interoperabilidad es un requerimiento SOA y que debe tratarse adecuadamente. Por ello se debe conocer la relación de un servicio entre consumidores y proveedores antes de hacer cambios sobre ellos.

- Integrar todos los servicios a SOA a la vez. Es recomendable comenzar relevando qué sistemas están funcionando y cuáles no, cuáles son los que aportan más valor y qué oportunidades de negocio se deben perseguir. Una vez listo esto, se debería confeccionar un plan que cubra la migración de todos los sistemas a SOA, proceso que se efectuará de forma incremental. Lo ideal es migrar o reconstruir las aplicaciones que darán beneficios de forma inmediata, por ejemplo las aplicaciones que todavía no están funcionando. Luego, utilizando un adaptador se puede colocar un simple "wrapper" sobre aquellas aplicaciones que ya están funcionando y aportando valor. Lo importante es asegurarse que la estrategia SOA se enfoque en mejorar los procesos de negocio que aportan más beneficios.
- Clonar aplicaciones. Esto significa que si en el código de la aplicación original había una falla entonces la misma se replicará, si se hicieron modificaciones durante este proceso difícilmente se hayan documentado, se tendrá infraestructura, mantenimiento y desarrollo redundante, y por último cada cambio en código que se produzca impactará en cada cliente que lo use y antes de hacer el deploy, durante las pruebas, se incrementará el trabajo, y en caso de encontrar una falla se deberá replicar la solución. Para evitar estas situaciones es recomendable tener un conjunto de aplicaciones centrales reusables de forma global a través de aplicaciones y procesos de negocio, incrementando la eficiencia y reduciendo el costo de mantenimiento.
- Uso de WSDL no gobernado podría caer en un consumo masivo de determinados servicios, compartiendo, quizás, datos que no deberían verse sino a determinados consumidores, en caso de presentarse una falla, no se tendría el control de quiénes estarían siendo impactados, etc. Por se recomienda:
    - identificar los consumidores y proveedores que dependen del servicio,
    - asegurarse de que se han tomado las medidas de seguridad pertinentes, se han reforzado las políticas de seguridad y que las reglas de negocio son aplicadas,
    - haber validado que el servicio que se está ejecutando y conoce los objetivos para los cuales fue diseñado está diseñado,
    - asegurarse que el servicio incluye los mecanismos de advertencia y capacidad de descubrimiento, de forma que sepa inmediatamente cuándo una operación falla.
- La validación externa de esquemas para evitar ataques puede inducir mayor vulnerabilidad si las validaciones radican en una determinada ubicación que un hacker pueda localizar y desviar hacia otra haciendo que las transacciones fallen. Una forma

sugerida para prevenir esto es dando los esquemas a quienes contactarán al servicio, luego poner la validación en una tabla interna securizada por las defensas de ambos perímetros y por seguridad end-to-end.

- Medir el éxito de SOA por la cantidad de servicios. Para medir esto es mejor centrarse en el reuso a lo largo de los servicios web, conocer el valor de un servicio cuantificando cuántas veces es utilizado, cuántos procesos son soportados, por lo tanto el número de ítems reutilizados. También se puede saber cuántos costos se está aprovechando debido al reuso, incluyendo el tiempo de diseño, de desarrollo y pruebas.

4.11 Desafíos SOA

Pierre Bonnet indicó que para alcanzar los desafíos que plantea SOA hoy día y crear valor en el dominio de los negocios se deben dar tres condiciones:

- *Crear una cadena de agilidad ACMS ó Agility Chain Management System (Sistema de Manejo de la Cadena Ágil): concretar las prácticas ágiles que se vienen estudiando teniendo como estrategia los procesos, reglas y administración de datos de referencia (master & reference data management).* La articulación de SOA con ACMS provee una arquitectura SOA más flexible.
- *Clarificar el governance en S*OA. En la llamada Cosmetic SOA, no hay mucho que gobernar dado que todo está basado en los sistemas back-end. En el contexto Re-engineering SOA todo toma una nueva dimensión. Destaca que lo importante es modelar los nuevos sistemas de información adecuadamente y efectivamente, para mantener y ajustar constantemente sus esquemas (blueprints). Si se tiene en cuenta una solución de respositorio y registro, no se habla de la conexión entre los lenguajes específicos del dominio y las herramientas como UML y IDE. La metadata puede ser expresada en modelos UML pero esta metadata no abre camino a los repositorios fácilmente.
- *Desarrollar una metodología que incluya Re-engineering SOA* basada en principios de ACMS que provea líneas guía para administrar los datos del negocio, casos de uso, procesos una arquitectura lógica (SOA), etc. Y que sea apoyada con los proveedores de infraestructuras. Allí plantea el uso de la metodología OpenSource que están desarrollando[61] donde la gente se sienta cómoda de hacer o sugerir modificaciones.

[61] http://www.praxeme.org/

Según Galen Gruman, en Cuatro pasos para transformar tu negocio (Four steps to making your business transform en inglés)[62], los desafíos SOA son los siguientes cuatro:

- *Desplegar en piezas pero creando un plan a largo plazo*. Creando la arquitectura y desplegar los servicios en fases, enfocándose quizás en un dominio de aplicación a la vez o eligiendo proyectos basados en la urgencia del negocio. Efectuar esto de forma incremental. La arquitectura involucrará identificar todos los procesos de negocio, las interacciones entre ellos, el flujo de la lógica de negocio y los datos para ejecutarlos. La identificación de éstas piezas, permitirán conocer las funciones comunes que pueden ser estandarizadas sobre todos los procesos y también, identificar la confianza, seguridad y administración de requerimientos.
- *Tomar seriamente el governance SOA* debido a que SOA involucra crear y administrar procesos IT como soporte a los procesos de negocio y el governance es crítico.
- *Poseer un staff profesional capacitado* ya sea en los procesos de negocio como en los servicios para desarrollar aplicaciones.
- *Aplicar también, los principios SOA a los datos*. Muchos servicios residiendo en múltiples aplicaciones pueden combinarse para ejecutar los procesos de negocio. Si cada servicio usa fuentes de datos diferentes, o incluso la misma fuente pero de diferentes formas, el resultado podría no ser el esperado.

Por último, según Eric Roch (en SOA Benefits, Challenges and Risk Mitigation[63]) SOA evoluciona con la maduración constante de estándares, nuevas ofertas de software y proveedores de soluciones. A su vez, indica que el riesgo tecnológico de SOA es desafiantes debido a los siguientes factores:

- Temprana adopción y evolución de tecnologías de soporte.
- El cambio organizacional es necesarios debido a que SOA cruza los límites de los sistemas.
- La arquitectura abarca toda la empresa incluyendo sistemas heterogéneos.
- La infraestructura es distribuida y requiere alta disponibilidad y escalabilidad.
- La metodología del ciclo de vida del proyecto requiere cambios debido a las dependencias complejas, patrones de diseño específicos SOA, y el impacto de los cambios en la infraestructura y los usuarios.
- El aseguramiento de calidad es más complejo debido a que los servicios se encuentran distribuidos, poseen muchas interfases, requieren nuevos ambientes de testing y herramientas de testing basadas en mensajes.

Luego indica los pasos para mitigar éstos riesgos:

[62] http://www.techworld.com/features/index.cfm?featureID=2574&printerfriendly=1
[63] http://blogs.ittoolbox.com/eai/business/archives/soa-benefits-challenges-and-risk-mitigation-8075

- Desarrollar un programa SOA con respaldo ejecutivo, objetivos en términos del negocio y con destino ROI (retorno de inversión).
- Establecer un Integration Competency Center (Centro de Integración Competente) integrado por un equipo que conozca de varias tecnologías.
- Examinar las arquitecturas y metodologías en uso y ajustarlas a SOA (OOA/OOD ágil con entregables específicos de SOA y patrones.
- Establecer un repositorio y políticas de governance para artefactos reusables como ser especificaciones de interfases (entregables de diseño), esquemas y definiciones de interfases (WSDL).
- Desarrollar una arquitectura de referencia SOA basada en patrones de diseño, herramientas útiles y buenas prácticas que definan lógica y físicamente SOA.
- Establecer un plan de entrenamiento para el staff del Competency Center.
- Adquirir herramientas de testing basadas en mensajes.
- Involucrar operaciones de soporte lo antes posible y desplegar herramientas de monitoreo y testing para la infraestructura SOA.
- Desarrollar una estrategia SOA y mapa de ruta basado en los valores del negocio, riesgos y efectividad de procesos de negocio.
- Hacer el pasaje a SOA de forma incremental e iterativa agregando servicios que aporten utilidad y valor al negocio.

4.12 Costo - Beneficio.

Para analizar la relación costo-beneficio se revisarán algunas encuestas efectuadas en 2005, 2006 y 2007. El objetivo es estudiar el comportamiento de las empresas ante la implementación de SOA desde 2005 hasta la actualidad, tratando de determinar cuál es la tendencia a futuro.

La empresa AMR [AMR-REP] efectuó en 2004 una encuesta que indicó que la mayor parte de los encuestados recién estaban comenzando las investigaciones acerca de los servicios web viéndolos como una gran promesa pero difíciles de diseñar, hacer su deploy y administrarlos. En 2005 efectuaron otra encuesta, esta vez ampliando el horizonte a SOA. Se encuestaron a 134 compañías que usaban SOA (generalmente concernientes a manufactura y servicios con planteles de más de 1000 empleados). Los encuestados eran IT Managers (CIO, vicepresidentes de IT, Directores IT, etc.). La encuesta arrojó las siguientes conclusiones:

- El 21% de los encuestados ya tenían al menos un deploy en SOA y el 33% tenían planificado implementar en los siguientes 12 meses.
- El mayor desafío IT fue la integración y el ensamblaje de componentes de software de varios proveedores a gran escala.
- Aquellos que planeaban hacer deploy de SOA esperaban poder reconfigurar los procesos de negocio más fácilmente que con los métodos tradicionales de integración.
- Se obtuvo gran carga de complejidad luego de la implementación de SOA si bien el objetivo es que los procesos de negocio sean configurables. Aquellos que hicieron deploy de SOA encontraron que las aplicaciones de software todavía no estaban modularizadas para trabajar en SOA, que los servicios de software modulares necesitan un repositorio común para manejar los servicios, que la capa de orquestación con BPM era necesaria para facilitar las relaciones entre los servicios, y que eran necesarios procesos de administración de cambios IT con el fin de garantizar la disponibilidad, confianza y escalabilidad de los servicios. La encuesta indicó que se planeaba invertir más en frameworks BPM, ESB y Master Data Management.
- El 65% de las compañías que en ese momento usaban SOA gastaron menos de 1 millón de dólares en relación con el 2004. En 2005 el 60% de ellas planeó incrementos de un 17% en promedio para el 2006.
- El 44% de los deploys SOA estaban enfocados principalmente en procesos de negocio internos, con integración de aplicaciones y mesas de ayuda IT. Luego la orientación al cliente fue la siguiente más popular elección para los proyectos SOA con un 32%, seguido por un 21% para la orientación a los proveedores.
- Los componentes SOA más empleados fueron los servicios web, frameworks para portales y servidores de aplicaciones. Producto de la falta de tecnologías estándares, la forma más común de implementar SOA fue mediante la compra de soluciones propietarias de determinados proveedores.

Del sitio InfoQ[64] se tomó la referencia a una encuesta realizada por BEA[65] en 2006. BEA encuestó en Norte América y Europa a 151 personas que desempeñan cargos importantes en cuanto a la toma de decisiones e influencias, cuyas empresas poseen un rédito mayor al billón de dólares y tienen al menos un programa piloto SOA, con el fin de estudiar la relación costo- beneficios de SOA[66]. Como resultado se pudo notar que:

- El 46% de las compañías intentan hacer deploy de 1 a 50 servicios dentro del año, el 24% de 51 a 100 y el 15% de 101 a 200 servicios.

[64] http://www.infoq.com
[65] http://www.infoq.com/BEA
[66] http://contact2.bea.com/bea/www/soa_mom/gcr.jsp?PC=%2059AMEAXXOIEM

- El 39% dijo que la justificación SOA impide el comienzo o extensión de proyectos SOA, y de ellos las razones fueron: falta de confianza en un ROI lo suficientemente alto y aseguramiento de fondos.
- En las empresas europeas el primer movilizador fue el ahorro de costos mientras que en Norte América fue la agilidad y el tiempo en los negocios.
- Más de la mitad de las organizaciones a las cuales pertenecen los encuestados han gastado más de 1 millón de dólares en SOA.
- El 40% de los costos listados para justificar SOA fueron ubicados en infraestructura de servicios.

El mayor gasto se correspondió con los ESB. A continuación le sigue gastos en seguridad y en servicios de datos.

En una de las encuestas llevada a cabo por la empresa IBM[67] en 2007, el 75% de los encuestados respondió que el la primera razón para implementar SOA es el buscar soluciones para nuevos problemas del negocio, mientras que el otro 25% para dar solución a problemas ya existentes. Luego, más de la mitad de los participantes respondió que sus presupuestos SOA para el año 2007 se incrementaron entre un 10 y un 20% en comparación con el año anterior y por último la encuesta indicó que más del 50% respondió que tienen menos del 25% del conocimiento necesario que necesitan sus compañías para el uso de SOA.

Tom Jowitt[68] en referencia a una encuesta realizada por Aberdeen Group realizada en 2007 a 400 empresas ("*SOA Middleware Takes the Lead: Picking Up Where Web Services Leaves Off*") publicó las siguientes conclusiones:

- Las organizaciones IT que efectuaron inversiones en infraestructura SOA (ESB, repositorios, registros de servicios, etc.) han logrado obtener: reducción de los costos a lo largo del ciclo de vida de las aplicaciones y un mejor nivel de satisfacción de sus usuarios.
- El 100% de los encuestados indicó una reducción de los costos de desarrollo de las aplicaciones y un 72% una reducción en los costos de mantenimiento de aplicaciones.
- El 71% usa SOA para simplificar o estandarizar sus infraestructuras de desarrollo.
- Un 76% ya tienen desplegados uno o más ESB.
- El 61% volvieron a instruir a sus equipos de desarrollo.

Por otra parte, el director de Aberdeen Group indicó que las organizaciones que tienen reportados casos de éxito consideraron necesarias las inversiones en infraestructura

[67] http://redmondmag.com/news/article.asp?EditorialsID=8801
[68] http://www.computerworlduk.com/management/it-business/it-organisation/news/index.cfm

de nivel empresarial. Por otra parte, indicó que se han desarrollado aplicaciones silos, aún cuando poseen una arquitectura de servicios web y SOA, todavía están sufriendo de la falta de una infraestructura de mensajería que abarque toda la empresa.

Para ampliar este concepto, cuando se habla de aplicaciones silo, se refiere a aplicaciones auto contenidas en un silo donde todos los servicios son provistos por código dentro de éstos. A esto se llegó debido a que por décadas muchas aplicaciones fueron desarrolladas como punto de soluciones para automatizar requerimientos departamentales. Como resultado, el código compartido entre aplicaciones silo es poco y a su vez, desconocido, con lo cual muchos servicios comunes son implementados redundantemente en múltiples silos. Esto también provee soporte escaso para procesos end-to-end como por ejemplo, el procesamiento de órdenes de compra o interacciones con proveedores. Por otra parte, las aplicaciones silo son difíciles de integrar debido a que cada silo generalmente tiene servicios separados e incompatibles.

En relación a las encuestas estudiadas, se puede decir que existe una tendencia marcada en el incremento en las inversiones sobre SOA, producto de los beneficios que trae aparejado SOA como arquitectura solución para empresas que buscan la agilidad, estandarización y flexibilidad y adaptación a los cambios en sus procesos de negocio. A continuación se incluye una lista de beneficios SOA que permiten la reducción de los costos dentro de una organización.

- Reutilización de servicios. SOA evita designar gran parte del presupuesto empresarial en el desarrollo desde cero, de nuevos sistemas que realicen las funcionalidades que proveen los sistemas legacy mediante la creación de interfases. Por otra parte SOA se basa en estándares (XML, SAML, SOAP & UDDI) que permiten desarrollar una arquitectura modular y por consiguiente el compartir y reusar servicios [SOAWP-HP].
- Integración de sistemas, una vez que una aplicación es parte de SOA puede ser fácilmente accedida por otra aplicación y generalmente sin implicar cambios en ésta. Esto permite una ganancia sustancial en términos de desarrollo y costos de integración [SOA-IMPLEM]. Por otra parte, el bajo costo de integración fue una de las principales motivaciones de SOA pudiéndose lograr mediante un diseño que tenga bajo acoplamiento, encapsulamiento y el reuso de servicios pueden ser modificados y corregidos sin necesidad de modificar sus interfases[69].

[69] www.ftponline.com/ea Rules Evolution in Service-Oriented Architecture

- Debido a que SOA permite eliminar redundancias, se necesitaría menor cantidad de servidores, menos licencias y menor mantenimiento implicando esto la reducción de costos[70].

Como también se vio en las encuestas anteriores, la adopción a SOA implica un incremento en los costos. Esto se debe a que las empresas deben capacitar a sus profesionales, invertir en seguridad y a que debido a la falta de estándares, mayormente se opta por usar herramientas de integración propietarias, como son los ESB o BPM. Si bien las encuestas marcaron una alta necesidad de incorporar este tipo de herramientas que permitan la orquestación de servicios, las empresas quizás podrían evitar parte de este gasto mediante la utilización de otras herramientas que automaticen parte del desarrollo y que sean Open Source (como ser Eclipse[71] que permite el desarrollo de una aplicación bajo SOA al igual que cualquier otro tipo de aplicación).

En relación con lo comentado anteriormente, también existe una metodología Open Source llamada Praxeme[72] cuyo co-autor es Pierre Bonnet, pensada para la construcción de SOA. Pierre plantea el surgimiento de ésta metodología como forma de afrontar las necesidades futuras elevando el nivel de madurez SOA. Actualmente el nivel es bajo y se lo llama "Cosmetic Maturity". Este se produce por la habilitación de servicios o "service enablement", es decir agregando cada vez más capas delante de los sistemas back-end provocando mayor complejidad en los mismos. El objetivo es llegar a un nivel de reingeniería o "Re-engineering" que permita reemplazar aquellos sistemas que ya no sean mantenibles.

Cabe también aclarar que todavía existe un amplio porcentaje de empresas que no han llegado a la fase de producción.

Joe McKendrick[73] en referencia a otra encuesta realizada por Evans Data[74] indica que la mayor parte de las organizaciones se encuentran en la transición de JBOWS(Just a Bunch Of Services ó sólo un puñado de servicios) a SOA y además todavía no poseen las piezas generalmente necesarias para construir SOA (entre otras menciona el governance y registros de servicios).

Esto indica que las implementaciones SOA todavía no tienen todavía el nivel de madurez necesario como para abarcar todos los factores posibles de éxito o fracaso

[70] http://www.iona.com
[71] www.eclipse.org
[72] http://www.praxeme.org/
[73] http://blogs.zdnet.com/service-oriented/?p=934
[74] http://www.evansdata.com/

relacionados, así como tampoco todos los costos y las ganancias que ésta pueda aportar si bien se mencionaron explícitamente factores que indican reducciones en los costos.

4.13 Electronic Data Interchange

EDI que significa Intercambio Electrónico de Datos, surgió en DLI (Minessota Department of Labor and Industry) en 1993.

Consiste en un conjunto de estándares para la estructuración de información para que sea electrónicamente intercambiable entre y dentro de los negocios, organizaciones, entidades gubernamentales y demás. Es considerado una representación técnica de conversación entre dos entidades. EDI incluye la transmisión, flujo de los mensajes, formato de los documentos y el software empleado para interpretarlos[75].

Los estándares EDI fueron diseñados para ser independientes de la tecnología de comunicación y del software.

EDI ofrece un tipo de comunicación más sencilla y barata mediante el intercambio de información estructurada capaz de facilitar la integración entre organizaciones remotas a diferencia del envío de documentos mediante un sistema de correo postal. Para que esto EDI requiere de una infraestructura tecnológica que incluya procesamiento de datos, manejo de datos y capacidades de trabajo en red, permitir la captura eficiente de datos de una forma electrónica y retención de datos, control de acceso y una transmisión eficiente y confiable entre los sitios remotos.

Los elementos que involucra son[76]:

- El uso de un medio de transmisión electrónico: redes locales ó públicas (Internet).
- Mensajes estructurados, formateados basados en acuerdos estándares de forma que el mensaje pueda no solo ser transmitido sino también interpretado y validado de acuerdo a un estándar.
- Entrega rápida del mensaje desde en transmisor hasta el receptor.
- Comunicación directa entre aplicaciones.

Por otra parte, existe una fusión de dos tecnologías XML y EDI desarrollada con el fin de extender las funcionalidades de EDI para el intercambio de diferentes tipos de información utilizando la sintaxis XML. Lo que se busca con esto es definir un nuevo marco para el desarrollo del comercio electrónico. XML/EDI es la unión de cinco tecnologías donde

[75] http://en.wikipedia.org/wiki/Electronic_Data_Interchange
[76] http://www.anu.edu.au/people/Roger.Clarke/EC/EDIIntro.html

cada componente aporta su tecnología. Los componentes que permiten la transmisión de los datos y el conjunto de reglas lógicas para interpretarlos son [XML/EDI]:

- XML que proporciona la base,
- EDI todo lo desarrollado hasta el momento,
- Plantillas incluidas dentro del XML que son reglas para que se pueda realizar el intercambio de información,
- Agentes creados principalmente con tecnologías Java o ActiveX que interpretan las plantillas para realizar el trabajo necesario e interactúan con las transacciones y el usuario creando nuevas plantillas para cada tarea específica y
- Directorios Globales de Internet que brindan la base semántica para las transacciones comerciales y apoyan a los agentes para realizar de forma automática y correcta la referencia a las entidades.

En Wikipedia[77] se pueden encontrar ejemplos de aplicaciones compatibles con EDI, como por ejemplo openXpertya[78] que es un ERP (Enterprise Resource Planning) open source adaptado para la legislación y el mercado español e hispanoamericano. OpenXpertya[79] provee una solución integral para la empresa englobando ERP y CRM, con integración de servicios en línea de B2B o B2C (en función del tipo de cliente final) e incluso B2E (servicios internos) y con soporte de exportación de datos (enlaces) al estandard EDI (intercambio electrónico de información entre empresa: facturas, albaranes, pedidos: EDIFACT, estandard mundial de la ONU) y con posibilidad de trabajar con cubos multidimensionales OLAP.

Por último, en [B2B-TRENCH] se muestra mediante porcentajes la proyección del avance de la tecnología EDI, Stateless XML y Stateful XML. Indica que si bien EDI fue capaz de crecer cuando no había otras tecnologías posibles siendo la primera tecnología con tipo de integración B2B, considera que tiene limitaciones en ambientes altamente cambiantes donde se requiere una tecnología más flexible. Por eso la tendencia muestra en primer lugar a las alternativas B2B basadas en XML pero concluye que quizás no vayan a reemplazar EDI en su totalidad.

4.13.1 Ventajas[80]

[77] http://es.wikipedia.org/wiki/Intercambio_electr%C3%B3nico_de_datos
[78] http://www.openxpertya.org/
[79] http://www.openxpertya.org/index.php?option=com_content&task=view&id=3&Itemid=4
[80] http://www.doli.state.mn.us/index.html

- Mejora el rendimiento. Los datos entregados como EDI son recibidos, procesados y conocidos en unas pocas horas en comparación con el sistema tradicional de correo postal.
- Ahorra tiempo. Ahorra el tiempo que insume el tratamiento de un papel desde que sale del remitente hasta el destinatario. Además evita los posibles errores y llamadas de confirmación de datos que pudiesen producirse al no tener la información precisa a enviar.
- Ahorra gastos. Si bien existen costos iniciales superiores a un tipo de envío tradicional de documentos, éste se recupera una vez que el sistema está en funcionamiento donde se requiere menor cantidad recursos y de personal para la administración de los mensajes a enviar.
- Reduce la cantidad de veces que determinados datos deben ser introducidos en varios sistemas de computadoras. Permite a las transacciones que efectúan el envío verificar y validar de forma inmediata sobre el receptor.
- Incrementa la flexibilidad debido a que el envío de datos puede efectuarse en cualquier momento del día. Las entregas pueden ser pautadas para determinados momentos donde los recursos de computación están siendo menos utilizados.
- La comunicación es más sencilla, veloz y barata.

4.13.2 Desventajas

Las empresas que requieren utilizar EDI como forma de intercambio de mensajes deben hacer un análisis exacto de cómo van a trasladar sus datos empresariales hacia los estándares EDI predefinidos. Esto hace que el proceso de implementación EDI sea lento, difícil y costoso.

Según el Director de Información de Sun Microsystems Bob Worrall en 2007 indicó que EDI provee controles de acceso a aplicaciones para seguridad y cumplimiento normativo. Considera que este es un asunto delicado dado que para un cierto número de compañías de alta tecnología que recientemente se han sumado a la tendencia de SOA, cuyos servicios Web a menudo tienen problemas al suministrar esta clase de seguridad pasada de moda. Creo que es muy comprensible. Los ambientes basados en la Web que vinculan servicios a través de múltiples aplicaciones, a menudo sacrifican los controles de acceso por el acceso general a las aplicaciones[81].

[81] http://www.sun.com/emrkt/innercircle/newsletter/latam/1006latam_sponsor.html

4.13.3 EDI y SOA

Desde el surgimiento de las aplicaciones de negocio, las empresas se han encontrado con la necesidad de integrar sus sistemas de forma tal que los usuarios puedan acceder de forma rápida a sus funcionalidades y a los datos de negocio. Esto se puede lograr mediante la implementación de tecnologías de información pudiendo utilizar varios diseños, desde las rígidas interacciones EDI extremo-a-extremo hasta las ventas por Internet. Con el paso del tiempo y la constante actualización de la tecnología, como lo son los sistemas basados en EDI sobre Internet, las compañías pueden hacer disponible sus sistemas IT a clientes internos o externos. Sin embargo, no siempre los sistemas resultantes son lo suficientemente flexibles como para cubrir todas las necesidades de los negocios.

Para lograr la integración de aplicaciones de empresas se debe tener en cuenta EAI y Business to Business (B2B). A continuación se da un detalle breve de cada uno.

EAI consiste en compartir sin restricciones datos y procesos de negocio sobre cualquier aplicación que se encuentre conectada a la organización y recursos de datos. Idealmente se espera lograr esto sin tener que hacer demasiados cambios a las aplicaciones o a las estructuras de datos aunque su implementación puede significar el desarrollo de toda una nueva visión del negocio y sus aplicaciones, donde éstas puedan ser adaptadas y se busquen formas de reusar eficientemente lo que ya existe agregando nuevas aplicaciones y datos[82].

Se puede decir que EAI resuelve éstos temas de una forma más amplia que los middlewares. A diferencia de EAI, los middlewares presentan limitaciones, por ejemplo sólo proveen soluciones punto a punto, los costos involucrados en las conexiones son elevados y se orienta solo a los datos faltándole la orientación a los procesos, etc. EAI incluye la noción de reuso y la distribución de datos y procesos, la incorporación de los sistemas legacy a una nueva infraestructura de aplicación de empresas [EAI-LINTH].

B2B también es también denominado como comercio electrónico y originalmente significaba la facilitación de transacciones comerciales electrónicamente, normalmente utilizando tecnología como EDI para enviar electrónicamente documentos como pedidos de compra o facturas. Con el tiempo evolucionó a la compra de bienes y servicios a través de la web vía servidores seguros incluyendo servicios de pago electrónico. Esto aportó a las empresas el poder recibir mayor cantidad de ofertas, la despersonalización de las ventas

[82] http://searchsoa.techtarget.com/sDefinition/0,,sid26_gci213523,00.html

evitándose tratos de favor y el abaratamiento del proceso entre otros beneficios[83]. B2B involucra tanto los protocolos de comunicación como las relaciones entre socios para las transacciones de negocio. Entre las actividades necesarias para definir una arquitectura B2B robusta se encuentran: análisis y diseño de workflows, diseño de la arquitectura, posible uso de web services, estandarización B2B (ebXML, BPEL, etc.) y estándares EDI[84].

Por su parte, para implementar un B2B o EAI, se requiere una arquitectura estándar y flexible que soporte diversas aplicaciones y permita compartir datos. SOA es una de esas arquitecturas dado que entre otras cosas, unifica los procesos de negocio estructurando gran cantidad de aplicaciones como una colección ad-hoc de pequeños módulos llamados servicios. Estas aplicaciones pueden ser usadas por diferentes grupos de personas, dentro o fuera de la compañía, y se pueden construir nuevas aplicaciones a partir de otros servicios pudiendo estos exhibir mayor flexibilidad y uniformidad[85].

Según Bob Worrall, los usuarios vinculan servicios bajo una SOA, en lugar de depender de métodos más antiguos de vinculación como EDI, el cual se limita a vincular instituciones con aplicaciones[86].

[83] http://es.wikipedia.org/wiki/B2B
[84] http://www.theserverlabs.com/content/view/19/37/lang,es/
[85] http://en.wikipedia.org/w/index.php?title=Service-oriented_architecture
[86] http://www.sun.com/emrkt/innercircle/newsletter/latam/1006latam_sponsor.html

5. METODOLOGÍAS ÁGILES DE DESARROLLO DE SOFTWARE Y SOA

5.1 Introducción

Mientras que los estándares, la tecnología y los servicios web son importantes para lograr SOA, también hay que reconocer que esto no es suficiente para que este tipo de arquitectura sea un éxito. Aplicando una capa de servicios web por encima de las aplicaciones legacy u otros componentes, no se garantizan las verdaderas propiedades de SOA, como ser: alineación con el dominio del negocio, flexibilidad, bajo acoplamiento y reusabilidad. Por ello es de suma importancia tener una aproximación sistemática y extensa que tenga en cuenta los requerimientos de software y siga las buenas prácticas recomendadas [SOA-SURV].

Cuando se habla de metodología para el desarrollo orientado a servicios no se está hablando de una nueva metodología, más bien se construye sobre el proceso de desarrollo que sigue la Organización y además agregando actividades y artefactos específicos para el desarrollo de los servicios. Si bien podrían tomarse cualquier proceso como base, dada la naturaleza compleja y cambiante de los requerimientos del negocio en el contexto de las empresas actuales, es recomendable seguir un enfoque iterativo incremental, con fuerte enfoque en la generación de productos intermedios para poder obtener productos claves que provean visibilidad sobre el desarrollo [SOA-BPPH].

A continuación se estudiarán las metodologías de desarrollo de software aplicadas en Arquitecturas Orientadas a Servicios. Se indicarán sus características, las metodologías en las que se basan, las estrategias y prácticas que sugieren, sus debilidades y fortalezas.

Antes de comenzar con el estudio de las metodologías en SOA se dará una explicación del ciclo de vida de desarrollo en el contexto SOA y las estrategias de delivery debido a que todo esto será de utilidad para comprender las secciones siguientes.

5.2 Delivery lifecycle en SOA.

El ciclo de vida en proyectos SOA consta de un conjunto de fases necesarias para que pueda llevarse a cabo la construcción de los servicios involucrados en una determinada organización. En comparación con otros ciclos de vida, es más granular y continuo.

De acuerdo al nivel de detalle y al autor, el ciclo de vida puede constar de 3 o 6 fases. En la versión simple las fases son: Definición, Diseño y Deploy. Se utilizará como referencia de éste ciclo de vida [SOA-LFQA].

En la fase de **Definición** se define el servicio desde la perspectiva de los requerimientos del negocio. Incluye la comprensión de contratos, dependencias con otros servicios, especificaciones de versionado y fin de vida del servicio. En una siguiente vuelta de rueda en el ciclo de vida, la fase de definición podría convertirse en Refinación.

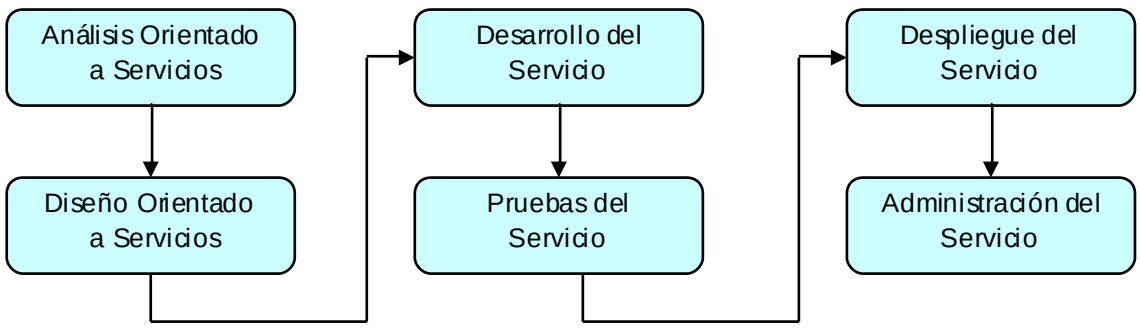
En la fase de **Diseño** se involucran los desarrolladores y personal de Control y Aseguramiento de Calidad. Se crean y componen los servicios o se impulsa el reuso. Se toman tanto especificaciones de seguridad y control de acceso como el aseguramiento de interoperabilidad y requerimientos de rendimiento. En esta fase, se pueden aprobar y publicar servicios.

En la fase de **Deploy**, finalizan las operaciones de IT y se administra la provisión, deploy y salida de operaciones de servicios. Se refuerza tanto la compatibilidad y contratos como la seguridad. Se monitorea el rendimiento para asegurar que el nivel de acuerdo del servicio es cumplido, y se propagan alertas y notificaciones cuando se requiera.

Debido a que se pueden producir cambios en los requerimientos, el ciclo de vida está en constante movimiento, de forma tal que en caso de producirse cambios, se ejecute el desarrollo del mismo, resultando en una nueva versión del servicio que deberá ser aprobada, publicada, provisionada, desplegada y administrada.

Por otra parte, Thomas Erl en [ERL-SOAC] define el ciclo de vida en mediante la descripción de 6 fases involucradas: Análisis, Diseño, Desarrollo, Pruebas, Despliegue y Administración. Si bien estas fases están presentes en un ciclo de vida de desarrollo de software tradicional, se incluye un breve detalle de cada fase dado que aporta un enfoque más específico para arquitecturas orientadas a servicios.

A continuación se incluye un diagrama con las fases a desarrollar en una orientación a servicios.



5.2.1 Fase de Análisis

En esta fase se determina el posible alcance de SOA determinando qué servicios se necesita construir y qué lógica deberán encapsular. Se hará un mapeo de las capas de servicios y los servicios individuales se modelarán como servicios candidatos que comprenderán la arquitectura orientada a servicios preliminar.

5.2.1.1 Objetivos.

- Definir un conjunto de servicios candidatos.
- Agrupar las operaciones de servicio candidatos en contextos lógicos que serán los servicios candidatos.
- Definir de forma preliminar los límites de los servicios verificando que no se solapen con los ya existentes.
- Identificar lógica encapsulada con potencial reuso.
- Asegurar que el contexto de la lógica encapsulada es apropiada para su uso esperado.
- Definir modelos de composición conocidos.

5.2.1.2 Proceso.

Los procesos descriptos son solo pasos suplementarios específicos de una solución con Orientación a Servicios. Tal como lo explica Erl no pretende suplantar a los procesos que ya ejecuta cada organización. Dependiendo de la estrategia de delivery de SOA, el análisis se puede aplicar en diferentes niveles y dependerá del tamaño del proyecto.

El proceso se divide en tres pasos:

Definir automatización de requerimientos del negocio. Se buscan y documentan los requerimientos de negocio. El alcance de este análisis se centra en la creación de servicios como soporte a una solución SOA. Esta información deberá ser suficiente para que se pueda definir la automatización de un proceso de alto nivel.

Identificar sistemas automáticos existentes. Si bien el AOS (Análisis Orientado a Servicios) no determina cómo será encapsulada o reemplazada la lógica de las aplicaciones legacy, debe asistir en el alcance de potenciales sistemas afectados.

Modelado de servicios candidatos. AOS introduce el concepto de modelado de servicios a procesos por los cuales las operaciones de servicios candidatos son identificados y luego agrupados en un contexto lógico. Estos grupos podrán tomar la forma de servicios

candidatos, luego ensamblados en un modelo de composición tentativo representando una lógica combinada de la aplicación orientada a servicios.

5.2.2 Fase de Diseño

Una vez determinado lo que se quiere construir, se necesita saber cómo hacerlo. El diseño orientado a servicios es una fase que comprende convenciones industriales, estándares y principios de orientación a servicios.

5.2.2.1 Objetivos.

- Determinar el conjunto de extensiones de arquitectura centrales.
- Fijar los límites de la arquitectura.
- Identificar los estándares de diseño requeridos.
- Definir los diseños de interfases de servicio abstractos.
- Identificar potenciales composiciones de servicios.
- Evaluar soporte para principios de orientación a servicios.
- Explorar soporte para características SOA contemporáneas.

5.2.2.2 Proceso.

En una primera instancia se establecen los procesos padres que comienzan con algún trabajo inicial. Esto lleva a una serie de procesos iterativos que gobiernan la creación de diversos tipos de diseños de servicios y por último el diseño de toda la solución.

1) **Componer SOA**. Uno de los principios en SOA es que cada instancia de SOA se puede componer de una única forma. Sin embargo algunas SOAs basadas en XML y WS-* permiten agregar extensiones en caso de ser requeridas.
Este paso consiste en:

a. **Elegir la capa de servicio** estudiando las capas de servicio candidatas producidas durante la fase de análisis y explorando las capas de servicios y escenarios de configuraciones. Cada capa abstrae un aspecto específico de la solución, ubicando una de las tareas identificadas. Esto facilita tener que construir servicios que acomoden el negocio, la aplicación y consideraciones de agilidad todas en una. Las tres capas de abstracción son:

   i. **Capa de Servicio de Aplicación**: provee funciones reusables relacionadas con el procesamiento de datos entre ambientes de

aplicaciones legacy u otras nuevas. Los servicios en esta capa pueden ser referidos como un servicio de aplicación simple y son responsables de la representación tecnológica y lógica de aplicación.

ii. **Capa de Servicio de Negocio**: introduce un servicio de forma individual representando la lógica del negocio. Pueden dividirse en distintas categorías de modelo de servicio dado que simplemente representan un grupo de servicios que expresan funcionalidad de tecnologías específicas. Así es como un servicio de aplicación puede utilizar un servicio, un wrapper de servicio u otra cosa.

iii. **Capa de Servicio de Orquestación**: aliviana la necesidad a otros servicios de manejar los detalles de interacción requeridos para asegurar que las operaciones de servicio sean ejecutadas en una secuencia específica. En esta capa, los servicios de procesos componen otros servicios que proveen funciones específicas independientes de las reglas del negocio y la lógica específica de los escenarios requeridos para ejecutar una instancia de proceso.

b. **Posicionar los estándares de SOA**. Evaluar los estándares y cómo serán implementados para soportar los requerimientos.

c. **Seleccionar las extensiones de SOA**. Requiere determinar qué características SOA se desea soportar. Esto permitirá decidir cuáles de las especificaciones WS-* deberán ser parte del ambiente orientado a servicios.

2) **Proceso de diseño de Servicios centrados en Entidades**. Antes de desarrollar un servicio se necesita tener definida su interfase. Esto es importante para establecer una alta estandarización de SOA y requerida para realizar características identificadas como parte de SOA contemporánea.

Reside en la capa de servicios de negocio y su objetivo es representar las entidades dentro de un modelo de negocio. Son soluciones construidas para que puedan luego ser reusadas por cualquier aplicación que requiera acceder o manejar información relacionada con una entidad. A continuación se mencionan y describen brevemente los pasos involucrados en esto.

i. Analizar servicios ya existentes. Es conveniente verificar si ya existe otro servicio que provea esta operación. En caso de existir verificar si cubre efectivamente una necesidad específica.

ii. Definir tipos de esquemas. En el caso de servicios centrados en entidades es beneficioso si se usa precisamente un esquema xsd que represente la información asociada con la entidad de servicio. A su

vez, estos esquemas pueden ser la base para la definición de los WSDL.

iii. Obtener una interfase de servicio abstracta asegurándose que cada operación candidata es genérica y reusable y mediante los datos definidos en esquemas xsd elegir el nombre de la operación, crear la interfaz (puerto) en el WSDL, publicar la operación y por último formalizar la lista de valores de entrada y salida requeridos para acomodar el procesamiento de cada lógica de operación.

iv. Aplicar principios de Orientación a Servicios.

    1. Reuso: el diseño es extendido para facilidad los requerimientos más allá de los identificados como parte de un servicio candidato.
    2. Autonomía: debido a que en general los servicios centrados en entidades son construidos para una capa de servicio padre y dado que se apoyan sobre la capa de aplicación para llevar adelante su lógica de negocio su autonomía inmediata es generalmente bien definida.
    3. Statelessness: por lo general los servicios centrados en entidades no poseen gran trato de lógica de flujo y por esos casos en los cuales aplicaciones múltiples o servicios de negocios necesitan ser invocados para llevar a cabo una operación, es preferible que el manejo del estado sea evitado tanto como sea posible.
    4. Descubrimiento, dado que es necesario asegurarse de que el diseño de un servicio no esté implementando lógica ya existente es necesario emplear servicios de descubrimiento y por otra parte es recomendable hacer que el servicio pueda ser descubierto suplementándolo, por ejemplo, con detalles de metadata mediante el elemento “documentation”.

v. Estandarizar y refinar la interfaz de servicio mediante una serie de tareas recomendadas como ser: revisiones de estándares de diseño y otras recomendaciones aplicadas apropiadamente.

vi. Extender el diseño del servicio a partir de analizar qué otros tipo de funcionalidades un servicio podría ofrecer. Para implementar nuevas funcionalidades se requerirá agregar nuevas operaciones y/o agregar nuevos parámetros a operaciones existentes pero teniendo en cuenta que a mayor cantidad de parámetros, se requerirá conocer más sobre

lo que el servicio hace para poder invocarlo. Las operaciones comunes para los servicios basados en entidades son: solicitar, actualizar, agregar o eliminar algo.

vii. Identificar procesamiento requerido y en caso de detectar que son necesarios servicios de aplicación, hacer un estudio que determine si ya existe otro servicio que brinda lo solicitado ó en caso contrario si el desarrollo del mismo debe ser efectuado y entregado como parte de la solución.

3) **Proceso de diseño de servicios de aplicación**. Los servicios de aplicación son los responsables del procesamiento que el negocio o las capas de orquestación les soliciten. La capa de servicios de aplicación es la abstracción de orientación a servicios del ambiente técnico de una organización.

   i. **Revisar servicios existentes**. Aquí es importante tener en estudiar si es necesario efectuar outsourcing de algunos servicios dado que generalmente están diseñados maximizando la reusabilidad.

   ii. **Confirmar el contexto** establecido por servicios de aplicación existentes. Las operaciones candidatas deben ser comparadas con servicios de aplicación existentes dado que puede llevar a encontrar que determinadas operaciones provienen de otros servicios de aplicación.

   iii. **Obtener una interfaz de servicio inicial** mediante el análisis de las operaciones candidatas.

      1. Asegurar que las particiones lógicas representadas por cada operación candidata sea apropiadamente genérica y reusable.
      2. Documentar los valores de entrada y salida requeridos para el procesamiento de cada operación candidata y definir la estructura de mensaje mediante esquemas xsd.
      3. Completar la definición del servicio agregando tipo de puerto y el mensaje con el tipo de esquema apropiado.

   iv. **Aplicar principios de orientación a servicios**.

      1. **Reuso**: asegurar que la operación fue diseñada para ser genérica y reusable.
      2. **Anatomía**: asegurar que la lógica de la aplicación responsable de ejecutar las operaciones de servicio no impone dependencias en el servicio o posee en si misma dependencias.

3. **Statelessness**: la mejor forma de promover un diseño de aplicación sin estado es llevando a cabo lo que más se pueda el diseño up-front.
4. **Descubrimiento**: un mecanismo de descubrimiento es útil para garantizar que el diseño no se solapa con la lógica provista por otros servicios de aplicación. También es recomendable la documentación del servicio incluyendo metadata.

v. **Estandarizar y refinar las interfases del servicio**. Se recomienda basarse en los mismos estándares y convenciones empleadas en la definición de otros WSDL de otros servicios.

vi. **Equipar el servicio candidato con funcionalidades especuladas** previamente verificando que no existan.

vii. **Identificar restricciones técnicas** como ser puntos de conexión, restricciones de seguridad, tiempo de respuesta y procesamiento, limitaciones técnicas de determinada aplicación.

4) **Proceso de diseño de servicios de negocio centrados en las tareas**. Aquí solo se tienen en cuenta las operaciones de servicio candidatas identificadas como parte del proceso de modelado de servicios.

i. **Definir el flujo lógico** usado para coordinar la composición de servicios comenzando por definir la lógica por cada posible escenario de interacción y debido a que ésta fase se efectúa en este momento, será necesario revisar los documentos de escenarios y concretarlos en modelos de interacción de servicios. Se debe documentar todos los posibles flujos con sus respectivos flujos de excepción. El objetivo también es obtener los mensajes que se intercambiarán y que están involucrados en el servicio.

ii. **Obtener la interfaz de servicio** por medio de:

1. Usar las operaciones del servicio de aplicación candidatas para definir las operaciones correspondientes.
2. Utilizar los diagramas de actividad y flujos lógicos para obtener las interfases de servicios.
3. Documentar valores de entrada y salida y el tipo de esquema asociado requerido para dicha operación.
4. Construir los WSDL con el tipo de puerto insertando la operación y agregar el mensaje xsd.

iii. **Aplicar principios de orientación a servicios**.

1. **Reuso**: si bien las oportunidades de reuso son raras también puede aplicarse en porciones del flujo lógico.
2. **Autonomía**: generalmente es dependiente de la autonomía de los servicios hijos
3. **Statelessness**: puede evitarse la persistencia de información de estado pasando con el uso del estilo de documentación de mensajes SOAP.
4. **Descubrimiento**: los servicios con centro en tareas pueden ser reusables y por lo tanto será necesario que puedan ser descubiertos por otros sistemas.
5. **Identificar procesamiento requerido**. Debido a que este tipo de servicios puede necesitar capa de servicios inferiores debe tenerse identificados todos los servicios de aplicación.

5) **Diseñar procesos de negocio orientados a servicios**. Se procede a crear una capa de orquestación que vincula nuestros servicios con la lógica de los procesos de negocio. Esto concluye con la definición ejecutable del flujo de trabajo lógico que se traduce en la creación de definición de proceso WS-BPEL.

5.2.3 Fase de Desarrollo

La elección del lenguaje de programación y el ambiente de desarrollo determinará la forma física del servicio y los procesos de negocio orquestados de acuerdo con sus diseños.

5.2.4 Fase de Pruebas

Las pruebas juegan un papel fundamental en SOA y debido a la naturaleza genérica y reusable los servicios deberán pasar por ellas antes del pasaje a un ambiente de producción. Deberán responderse preguntas como ser:

- ¿Qué tipo de consumidores podrán acceder potencialmente al servicio?
- ¿Pueden todas hallarse todas las aseveraciones de políticas de servicio?
- ¿Qué tipos de condiciones de excepciones puede potencialmente un servicio experimentar?
- ¿Cuán bien una descripción de servicio comunica la semántica del mismo?

- ¿Las descripciones de servicio alteran o extienden versiones previas?
- ¿Cuán fácil puede ser compuesto un servicio?
- ¿Cuán fácil puede la descripción de un servicio ser descubierta?
- ¿Es compatible a los perfiles WS-I requeridos?
- ¿Qué problemas relacionados al tipo de datos pueden aparecer?
- ¿Han sido mapeadas todas las actividades de servicio y composición de servicio?
- ¿Cumplen todos los servicios con estándares de diseño?
- ¿Incluyen encabezados SOAP los nuevos servicios y pueden sus consumidores comprenderlos?
- ¿Introducen los nuevos servicios requerimientos funcionales o de calidad de servicio que la arquitectura actual no soporta?

5.2.5 Fase de Deployment

Involucra de acuerdo a la plataforma tecnológica de desarrollo determinadas tareas de instalación, configuración de componentes distribuidos, interfases de servicio, y productos intermediarios en servidores de producción. Algunas cuestiones a tener en cuenta son:

- ¿Cómo serán distribuidos los servicios?
- ¿La infraestructura es la adecuada para cumplir los requerimientos de todos los servicios?
- ¿Cómo afectará a los servicios y aplicación la introducción de nuevos servicios?
- ¿Cómo serán posicionados y desplegados los servicios compartidos?
- ¿Cómo afectará la introducción de middleware requerido en ambientes existentes?
- ¿Qué características de seguridad y cuentas son requeridas?
- ¿Cómo serán mantenidos y monitoreados los sistemas legacy encapsulados con limitaciones de rendimiento y confiabilidad

5.2.6 Fase de Administración

Será necesario tratar cuestiones que incluyan:

- ¿Cómo será monitoreado el uso de un servicio?
- ¿Qué tipo de control de versiones será usado para gestionar los documentos de descripción de servicios?
- ¿Qué mensajes serán traceados y manejados?
- ¿Cómo serán detectados los cuellos de botella de rendimiento?

5.2.7 Estrategias de Delivery

Según [SOD-P&H], existen básicamente tres estrategias de delivery en arquitecturas orientadas a servicios, teniendo en cuenta la cantidad de análisis de front-end[87] (que es la parte del software que interactúa con el usuario) del dominio de negocio y el tratamiento de los sistemas legacy (ver Glosario) existentes: Top-Down, Bottom-Up y meet-in-the-middle de tipo ágil. La aplicación de las mismas no es excluyente, generalmente se las aplican en conjunto. A continuación una breve descripción de cada una de ellas.

Cuando se trata de **Top-Down**, primero se analiza el dominio de negocio y luego se descompone el negocio en procesos y subprocesos de negocio, y en casos de uso. Se identifican operaciones de los servicios desde las tareas del negocio [IBM-TERM]. Los casos de uso generalmente son candidatos a ser servicios. Está altamente unida a la lógica de negocio y de la cual se derivan los servicios requeridos en la organización. Normalmente se emplea esta forma, debido a que se basa en las funciones de la organización. Además la ventaja que tiene es que uno se puede asegurar que el servicio identificado está alineado con el dominio del negocio [IBM-TERM].

La estrategia **Bottom-UP** a diferencia de la Top-Down, esta aproximación analiza los activos de IT ya existentes como ser, aplicaciones legacy y otros sistemas, encontrando funcionalidades que puedan ser expuestas como servicios (teniendo la precaución de no exponerlas ciegamente de forma que no sean de tan poca granularidad y no estén alineadas con el negocio), y reusadas por otros [IBM-TERM].

La granularidad se refiere al nivel de abstracción de un servicio [BEA-DMOD].

Un servicio de *grano fino* ofrece funcionalidad discreta, como por ejemplo, un método basado en estándares para llamar a una API (servicio de acceso) o una forma de operar sobre una entidad de datos empresariales (servicio de información). Los servicios empresariales compartidos también suelen ser de grano fino, proporcionando operaciones empresariales comunes tales como cálculos financieros.

Los servicios de *grano grueso* proporcionan un mecanismo para acceder a funcionalidad empresarial de mayor complejidad, como por ejemplo, la incorporación de un empleado o el procesamiento de un pedido de un producto o servicio. Generalmente se ejecutan durante largo tiempo e implican la coordinación de la ejecución de servicios de grano más fino. Esto hace que su implementación sea más compleja.

87 http://es.wikipedia.org/wiki/Front-end_y_back-end

Por último, la tercera estrategia **Meet-in-the-middle** encuentra el balance entre la incorporación de principios de diseño de Orientación a Servicios en ambientes del negocio sin tener que esperar la integración de las tecnologías de web service en ambientes técnicos [SOD-P&H]. Por otra parte, se enfoca en la conciliación de necesidades (los servicios identificados por el análisis Top-Down) y lo que ya es provisto por los activos existentes en IT [IBM-TERM].

Requiere de la colaboración entre analistas de negocio, arquitectos de software y especialistas para aplicaciones legacy. Si bien es valuable realizar análisis Bottom-Up fuera del contexto de un proyecto específico, en la práctica, sin embargo el esfuerzo usualmente comienza con una aproximación Top-Down y la descomposición de uno o más procesos de negocio como parte de un proyecto. Luego, en este contexto, ocurre el análisis Bottom-Up intentando encontrar coincidencias para los servicios requeridos [IBM-TERM].

Como ejemplo de Top-Down existe IBM's Service Oriented Modeling and Architecture[88] (SOMA) que es una metodología que toma como base Top-Down. A su vez, utiliza un enfoque Bottom-Up, con el fin de analizar sistemas legacy y componentes existentes y luego realizar servicios definidos en el dominio de negocio y descubrir otros nuevos, que no hayan sido encontrados mediante Top-Down.

Por su parte, OASIS[89] SOA Adoption Blueprints, define una aproximación de descomposición de servicios y provee una notación para los servicios similar a los casos de uso de UML. Recomienda comenzar desde un nivel funcional (de negocio) examinando las áreas funcionales de negocio.

Según Eric Roch[90], es recomendable emplear una metodología híbrida que tome ideas de SOMA y OASIS de forma tal que cree servicios de alto nivel usando métodos funcionales y luego proveyendo más detalle solo para servicios estratégicos en el proceso de negocio y a nivel de caso de uso. De ésta forma, Roch asegura que se podrá lograr un catálogo de servicios publicados con más detalle para los servicios que serán creados tempranamente en la transformación SOA.

88 IBM's SOMA and the RUP update for SOA, http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/

89 OASIS Contribution, http://www.oasis-open.org/committees/download.php/15071/A%20methodology%20for%20Service%20Architectures%201%202%204%20-%20OASIS%20Contribution.pdf#search=%22OASIS%20SOA%20methodology%20%22

90 Eric Roch, http://www.ittoolbox.com/profiles/eroch

5.2.8 Metodologías SOA

En esta sección se estudiarán las metodologías aplicables en Arquitecturas Orientadas a Servicios. El objetivo de estudio de éstas metodologías es conocer cada una de ellas de forma tal de poder establecer una base, sobre la cual luego serán comparadas cada una de ellas, para determinar cuál podría ser más adecuada para el desarrollo de software en este tipo de arquitecturas.

Una de las recomendaciones clave en el desarrollo de arquitecturas SOA es adoptar y adaptar una metodología de desarrollo que permita identificar, diseñar y construir componentes SOA [SOA-REDBOOK].

Se puede decir que los proyectos SOA poseen mayor complejidad y requieren mayor grado de logística y comunicación que los proyectos de software típicos debido a que conviven y se interrelacionan equipos de desarrollo de diversos sistemas cuyos servicios se comparten y son accedidos entre sí. Para optimizar el desarrollo en proyectos con SOA deben considerarse los factores clave: una clara definición del proceso de desarrollo, incremento de las líneas de comunicación entre equipos de desarrollo que conocen acerca del negocio, un claro soporte y políticas de gobierno y por último y no menos importante una metodología de desarrollo formal [MITTAL-SOA].

Actualmente existen varias metodologías que pueden ser aplicables en SOA. La mayor parte de ellas se basan en las Metodologías Ágiles de desarrollo de software y otras en las tradicionales, tomando las mejores prácticas en cada caso.

Las metodologías que se describirán son:

- Agile.
- RUP.
- SOMA.
- XP.
- Repeatable Quality.
- SOUP.
- Lean Software Development.
- CBDI-SAE.
- SOAD.
- Arquitectura de Servicios de Steve Jones.
- BEA.
- SOAF.

En la sección 5.3 se describe cada una de ellas y en la sección 5.4 se realiza la comparación entre ellas.

5.3 Metodologías en detalle.

En esta sección se estudian las metodologías de desarrollo aplicables en SOA. Se indica sus características, ventajas, limitaciones y desafíos a futuro. Estos detalles servirán de base a la sección 5.5 donde se hace una comparación entre ellas en base a las experiencias observadas.

5.3.1 Ágiles.

Las prácticas ágiles ponen énfasis en el desarrollo iterativo e incremental, contando con integraciones continuas y desarrollo dirigido por pruebas (test-driven) de forma que se facilite el desarrollo de aplicaciones en SOA. Quienes construyen en ambientes SOA deben usar soluciones ágiles e iterativas para facilitar lograr el éxito [AGILE-SOA] debido a su gran conexión.

Cualquier responsable en SOA deben enfrentar dos desafíos básicos: el primero radica en cómo se diseñarán sistemas de forma que encajen adecuadamente entre los procesos de negocio, los objetivos del negocio y en las arquitecturas IT y el segundo es cómo construir una arquitectura de tal forma que ésta sea capaz de responder a cambios futuros [IBM-AGILE]. Lo cierto es que en los proyectos SOA que implementan procesos ágiles, los equipos podrán alcanzar de una forma efectiva estos desafíos [AGILE-SOA] y los siguientes:

- Tiempo para negociar y tiempo para evaluar, prioritarios para el equipo de desarrollo, ambos principios SOA y los procesos ágiles explotan la necesidad de entregar de forma iterativa e incremental requiriendo la presencia del cliente como conductor de los requerimientos trabajando en conjunto con el equipo de IT.
- Sin procesos o procesos débiles. Cualquier nuevo tipo de servicio son entregados, los equipos pueden cuantificar el valor de lo que han entregado y pueden unir el costo y beneficio en componentes de software individuales. Esto puede justificar el valor de adoptar nuevos procesos o herramientas asociadas.
- Pruebas como el hueco negro. Los procesos ágiles destacan sobre las pruebas tempranas y complementos de prueba frecuentes la habilidad de poder ejecutarlas de forma individual sobre servicios e interfases mientras se continúa con el desarrollo de las

funcionalidades. Esto mismo se verá en las compañías que adopten técnicas ágiles en sus proyectos SOA.

- Arquitectura fuerte. Para que los procesos ágiles satisfagan verdaderamente las necesidades de la empresa deben cumplir los requerimientos y el ambiente de software que la compañía soporta.

Tal como se mencionó en la sección "Agile y SOA", una de las limitaciones de Agile es que si no se posee el suficiente nivel de colaboración y comunicación entre los participantes, entonces el proyecto y las prácticas ágiles podrían fracasar.

Ver también 3.2.7 donde se listan algunas limitaciones de las metodologías ágiles que también serían aplicables si se emplean en proyectos con SOA.

5.3.2 RUP.

En secciones anteriores se estudió ésta metodología indicando sus características, fases del ciclo de vida de desarrollo de software, artefactos, hitos a cumplir, ventajas y desventajas. En esta sección se incluye un análisis de los beneficios y limitaciones que presenta esta metodología cuando se aplica a proyectos sobre SOA.

Tal como se describió esta metodología anteriormente, RUP es una metodología que se centra en los requerimientos, el modelado y la generación de documentos. Tal como se menciona en [SOA-RUPXP] un beneficio de ésta metodología en SOA es que sus prácticas favorecen a las Arquitecturas Orientadas a Componentes y con ello, la reusabilidad, abstracción de los procesos de negocio, flexibilidad y codificación con bajo acoplamiento y fina granularidad.

Según Eric Roch[91] en "RUP versus Agile for SOA Methodology", es posible crear una instancia liviana de RUP como la metodología para el diseño de entregas en SOA dado que la naturaleza de RUP es iterativa y con foco en lograr una arquitectura lo antes posible en el proceso. La clave del éxito es mantener los procesos livianos e iterativos. Roch[92] sugiere primero implementar una metodología como es SOMA y OASIS Blueprints para el descubrimiento de servicios y luego a un nivel más bajo utilizar RUP y UML.

[91] http://www.ittoolbox.com/profiles/eroch

[SOA-RUPXP] indica como desafío para esta metodología teniendo como objetivo su adaptabilidad a proyectos con SOA lo siguiente:

- Enfoque en el diseño.
- Concentrarse en el desarrollo de servicios.
- Considerar equipos de trabajo más numerosos.

Según [SOA-RUPXP], una de las principales limitaciones de RUP en la posible aplicación en proyectos con SOA es que RUP está más Orientado a los Objetos sobre todo en el diseño no siendo esto lo adecuado para el modelado de servicios que incluye muchos factores que éste paradigma excluye, por ejemplo a agregación de servicios, el hecho de que ignora la infraestructura sobre la que se basa, (protocolos, ubicación y tecnologías) y la administración de servicios. Otro motivo es que RUP no aporta gran agilidad en comparación con otras metodologías dado que incluye más actividades y artefactos en cada iteración.

Otro punto marcado por Roch[93] es que para las empresas que comienzan con un modelo bottom-up y luego avanzan a uno top-down RUP no es recomendable. RUP y UML son buenos para la captura en proyectos basados en entregables que puedan ser reutilizados como lo son los servicios. Los entregables soportan el reuso de servicios.

5.3.3 SOMA.

Las siglas SOMA provienen de Service-Oriented Modeling and Architecture que significa Modelado y Arquitectura Orientada a Servicios. SOMA describe un conjunto de productos y tecnologías de modelado agnósticas, actividades de análisis y diseño y técnicas para construir SOA.

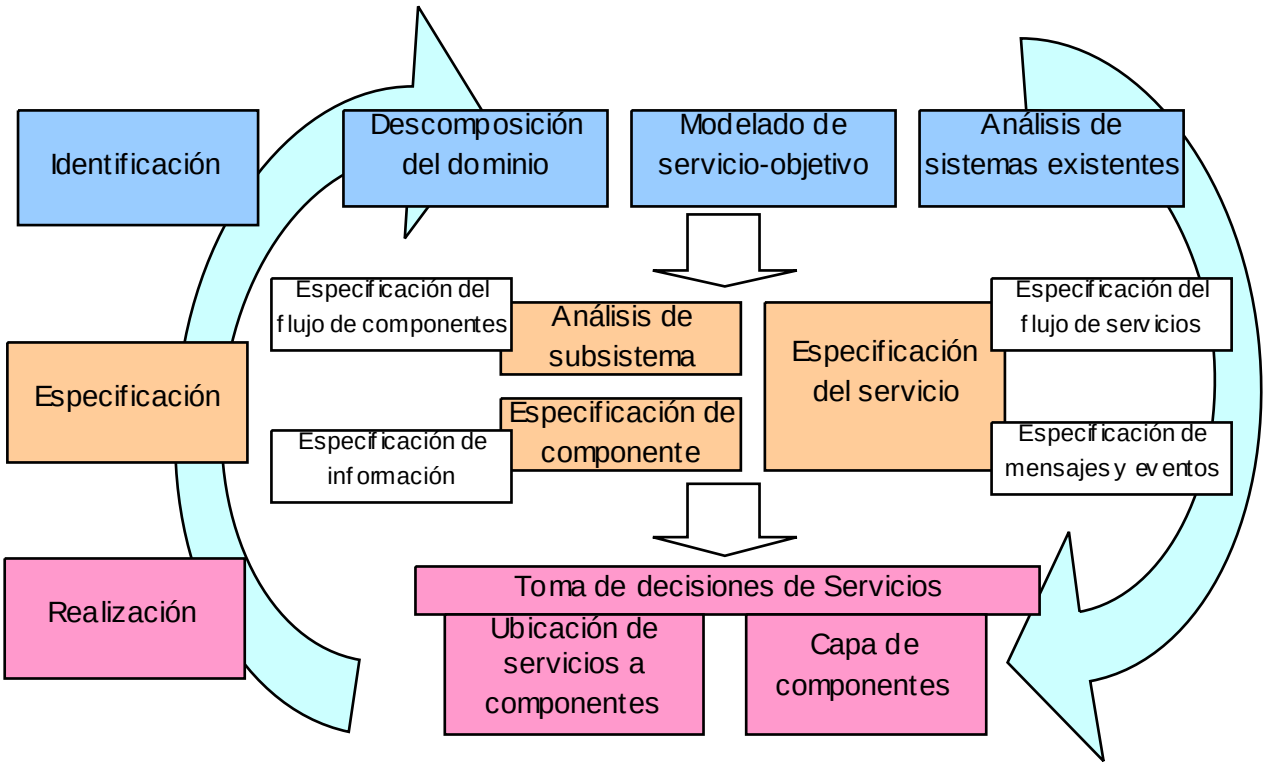
Identifica funciones relevantes a exponer como servicios de negocio SOA dentro del dominio funcional, para soportar a los procesos de negocio y dentro de los nuevos proyectos de aplicaciones.

Especifica su alcance y los requerimientos no funcionales, como por ejemplo la calidad de servicios. Toma decisiones para su realización y crea un modelo de servicio que incluyendo un portfolio de servicios.

El proceso se compone de tres fases: identificación de servicios candidatos y flujos, especificación de servicios, componentes y flujos y realización que involucra la toma de

[93] Agile: The SOA Hangover Cure. http://carlaugustsimon.blogspot.com/

decisiones. SOMA permite identificar antipatrones e indica la forma de evitarlos. A continuación, se incluye un diagrama que muestra las fases antes mencionadas. Por último se da detalle de las fases, patrones, antipatrones y qué acciones tomar para evitarlos [SOA-REDBOOK].



SOMA toma prácticas de RUP para la identificación de servicios y para ver cómo los procesos de negocio son efectuados a través de la ejecución de servicios.

Se ubica en la fase de incepción y elaboración de RUP antes mencionadas. Teniendo en cuenta RUP, se requiere un artefacto más, un modelo de servicios.

SOMA crea continuidad entre los negocios y las implementaciones de IT extendiendo las características de los negocios (por ejemplo, objetivo e indicadores de rendimiento claves) en el análisis IT y decisiones de arquitectura. El análisis y modelado realizado durante SOMA es agnóstico en cuanto a la tecnología y el producto, pero establece un contexto para tomar decisiones de tecnología y producto en procesos posteriores del ciclo de vida [SOMA-PPT].

a) Identificación de servicios:

- Modelado del servicios utilizando alguna de las estrategias de delivery en SOA.
- Descomposición del dominio: representa la aproximación top-down, donde el dominio del negocio es descompuesto en áreas funcionales las cuales cada una puede ser

descompuesta en procesos y subprocesos y casos de uso de negocio de alto nivel, posibles servicios candidatos.

- Análisis de capital existente: representa la aproximación bottom-up donde se analiza APIs, transacciones, y modelos de los legacy y aplicaciones empaquetadas como posibles servicios.
- Modelado de servicios-objetivo, aproximación meet-in-the-middle que relaciona los servicios con sus objetivos, subobjetivos, métricas de la empresa, etc.
- Análisis de subsistemas, expande sobre los subsistemas identificados durante la descomposición del dominio y especifica interdependencias y flujos entre ellos.

Antipatrones:

Síndrome de proliferación de servicios: se produce cuando hay tendencia a equiparar servicios web con SOA, cuando en realidad los servicios web son el punto de entrada a SOA.

Determinando el nivel más apropiado de granularidad de servicios es uno de los desafíos en el modelado de SOA. La regla es modelar tan grueso (coarse-grained) como sea posible. El objetivo es lograr el balance entre la granularidad fina y la gruesa. Se describen técnicas para determinar la granularidad apropiada de los servicios y minimizar la proliferación de servicios de muy fina granularidad.

Solución Silo:

Los servicios son identificados basados en aplicaciones aisladas más que en un foco más general de la empresa.

b) Especificación de servicios:

Se identifican y especifican componentes que serán requeridos para descubrir servicios, por ejemplo: identificación de reglas, servicios, atributos, dependencias y puntos de variación para cada componente, exploración de requerimientos no funcionales requeridos por servicios consumidores, especificación de mensajería, eventos, etc. y decisiones de manejo de estados.

Lo importante es determinar qué servicios serán expuestos. Algunas cuestiones que pueden ayudar a determinar esto son: trazabilidad, stateless (si el servicio debe poseer un estado o no), descubrimiento, reuso.

c) Ubicación del servicio: se asignan servicios a subsistemas identificados previamente.

Antipatrones:

Existe la tendencia de saltar directamente al desarrollo e implementación de servicios web sin una asociación clara con los componentes dueños. Esto ocurre en proyectos con falta de disciplina en arquitectura donde no se aplican ni se consideran los patrones.

La falta de modelado de servicios y técnicas de diseño resultan en violaciones serias a principios básicos como el diseño modular, ocultamiento de información, encapsulamiento y división en capas.

d) Realización de servicios: involucra la exploración de alternativas para realizar la implementación de servicios. Se identifican restricciones técnicas para asegurar la factibilidad técnica de realización de servicios en especial para los identificados durante el análisis. Cuando se trata de implementar un servicio se evalúa la posibilidad de construirlo o comprarlo.

e) Consideraciones de transformación: una implementación puede consistir en crear un wrapper a un sistema existente o a un servicio web pero en ciertos casos esto no es suficiente. También se puede componentizar un sistema existente que evita la tendencia de romper un sistema entero en partes y selecciona el subconjunto apropiado y lo transforma en componentización.

Antipatrones:

Chatty services: se da cuando los desarrolladores comienzan el reemplazo de APIs existentes por servicios web sin antes analizar los beneficios de SOA ni seguir sus principios de arquitectura. Se le llama Chatty Services dado que se comunican con pequeñas piezas de información, que puede llevar a un alto grado de degradación en el rendimiento.

Para evitar este escenario SOMA sugiere:

- En la fase de indentificación, usar meet-in-the-middle para asegurar que el modelo sea completo y tenga el grado de granularidad apropiado y asegure la trazabilidad desde un servicio a los objetivos del negocio y metas de la empresa.
- En la fase de especificación, la aplicación de “test litmus” considerando principios de diseño para tomar las decisiones apropiadas que involucran la exposición del servicio. Con el test Litmus se pueden interrogar las siguientes cuestiones [IBM-SOMA]:
    - ¿Es una función del negocio? Deshacerse de operaciones técnicas ó componentes de sistemas existentes.
    - ¿Es una interacción simple y sincrónica con IS? Dependencia en la implementación vista.

- Encapsulamiento: ¿Limita con algún componente de negocio? ¿Qué sucede si el servicio es provisto desde afuera?
- ¿Es reusable? ¿Qué pasaría si no lo fuera?
- ¿Es de alto nivel de granularidad?

- En la fase de realización, asegurar la factibilidad técnica de los servicios identificados a través de un análisis para validar luego las decisiones de exponer un servicio.

Un proyecto que aplique ésta metodología, podría considerar como beneficioso, los puntos que se indican a continuación.

- *Define para cada fase antes mencionada, los posibles antipatrones y la forma de evitarlos.*
- *Contempla los tres pasos fundamentales concernientes al desarrollo de servicios desde la identificación de servicios candidatos y flujos, selección de servicios, componentes y flujos que serán expuestos y las decisiones acerca de cómo serán implementados.*
- *Utiliza las buenas prácticas del análisis y diseño orientado a objetos, también UML como herramienta de modelado y UDDI para la publicación y descubrimiento de servicios.*

Si bien el análisis y modelado realizado durante SOMA es agnóstico con respecto a la tecnología y el producto, establece un contexto para tomar decisiones específicas de tecnología y productos en las últimas fases del ciclo de vida [SOMA-PPT].

5.3.4 XP.

Esta metodología ya se ha estudiado en secciones anteriores. Ahora se indicarán las ventajas y desventajas de aplicarla en desarrollos sobre SOA.

Según [SOA-RUPXP], muchas prácticas y beneficios de XP aportan valor sobre SOA. XP puede ser aplicable favorablemente en SOA debido que permite:

- Efectuar cambios de código en cada pequeña y rápida iteración. Este es un punto a favor teniendo en cuenta SOA debido a que convive con escenarios inciertos y cambiantes donde se prevén modificaciones constantes en el negocio, protocolos, código, etc.
- Favorece la comunicación del cliente tomándolo como parte activa en el proyecto. Esto es fundamental en proyectos SOA donde las reglas de negocio y procesos varían asiduamente y la comunicación permite una mejor descripción de las necesidades y requerimientos.
- Favorece la comunicación entre los desarrolladores proveyendo grandes mejoras en la productividad y mejores resultados al final del proyecto.

Un ejemplo de uso de XP en proyecto SOA es Digital Focus[94], una compañía de Virginia que usa XP y SOA en proyectos para Federal Home Loan Banks en la oficina de finanzas dado que encontraron en XP la forma de hacer las planificaciones a corto plazo con entregas periódicas empleando técnicas de desarrollo incremental como complemento a las prácticas en el desarrollo de modelos de procesos de negocios.

[SOA-RUPXP] indica como desafío para esta metodología teniendo como objetivo su adaptabilidad a proyectos con SOA lo siguiente:

- Desarrollar componentes para una arquitectura SOA.
- Concentrarse en el desarrollo para arquitecturas SOA.

Por otra parte, según [SOA-RUPXP], las limitaciones de XP en el desarrollo de aplicaciones SOA son las siguientes:

- Falta de diseño tanto de requerimientos como de sistema sabiendo que en dentro de la estructura de negocios existe gran cantidad de requerimientos que podrían causar fallas o producir una arquitectura con un diseño equivocado. En arquitecturas SOA donde los servicios forman los bloques de construcción, que debe soportar cambios constantes en el negocio, que cuenta con gran cantidad de variables y con la necesidad de que estos servicios funcionen correctamente bajo cualquier situación requiere un buen diseño de antemano.
- Falta de documentación o un modelo de diseño en XP es reemplazado supuestamente por las pruebas, la programación de a pares, y la disponibilidad de un cliente on site. Pero si bien las pruebas constantes pueden detectar fallas, algunas veces la falla en el diseño pueden ser detectada mucho tiempo después y por más que el cliente esté disponible y que se efectúe la programación de a pares, será requerida determinada especificación que permita guiar a los desarrolladores a través del proceso.
- El tamaño del equipo y la localización. Puede presentar problemas si se trata de un equipo distribuido geográficamente dado que sería prácticamente imposible la programación de a pares.

Según Eric Roch[95] en "RUP versus Agile for SOA Methodology", la falta de rigor en XP puede llevar a un incompleto e inconsistente proceso en relación al dominio de negocio, el modelo común de información (esquemas representando objetos de negocio comunes) y las

[94] http://www.digitalfocus.com/research/whitepapers_soa.php.

[95] http://www.ittoolbox.com/profiles/eroch

especificaciones de diseño de interfases. Debido a que estas entregas se tornan las bases de reusos futuros, la consistencia del proceso es crítica. Por otra parte, lo que se intenta evitar tener que mirar el código para determinar qué es lo que hace el sistema o tratar de descifrar un esquema XML para entender su semántica. Allí es donde un diseño de entregas mínimo toma importancia. Es mucho mas sencillo mirar una o dos páginas de un caso de uso para entender lo que el servicio hace y ver un diagrama de secuencia para entender las invocaciones del determinado servicio que tener que mirar el código.

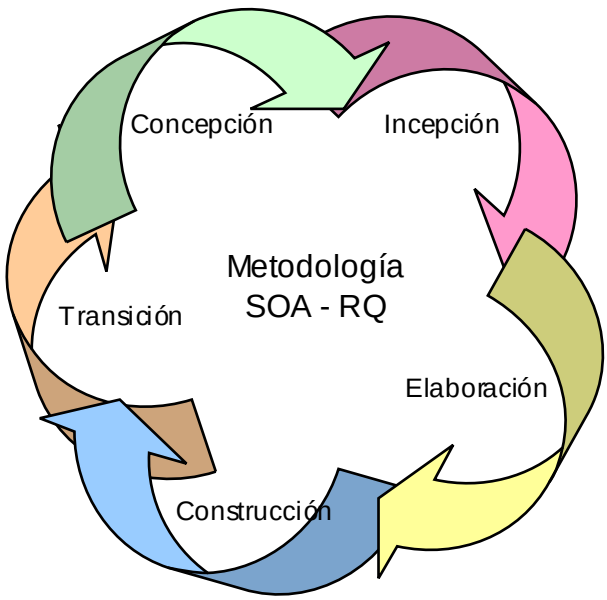
5.3.5 SUN Repeatable Quality (RQ).

A continuación se detalla la metodología SOA RQ descripta por [SUN-RQ].

Es una metodología adaptativa, basada en roles, iterativa e incremental para el desarrollo de proyectos sobre SOA. Emplea una combinación de estándares de industria, UML y RUP y técnicas XP. Guía el ciclo de vida SOA mediante la implementación de templates listos para usar e implementar en todas las fases del ciclo de vida de SOA.

Las iteraciones son guiadas por riesgos, diseñadas para ubicar las áreas de mayor riesgo tanto técnicas como de negocio. En cada una se entrega un componente ejecutable que será evaluado de acuerdo a los requerimientos del negocio.

RQ posee diferentes perspectivas para los analistas del negocio, el staff de operaciones y los representantes ejecutivos. Se lo denomina vista 4+1.



Fases en RQ. Fuente: [SUN-RQ]

Facilita la creación de una vista unificada haciendo audiencias a empresas que estén interesadas en las capacidades SOA. Para evitar que los usuarios no se involucren en el

ciclo de vida de la implementación SOA porque podrían ocurrir retrasos y no sentirse parte del de la solución final, RQ propone talleres estructurados tanto para personal técnico como para los de negocio, que favorecen el nivel de colaboración. A su vez provee artefactos que identifican, definen y documentan el proyecto. Estos ilustran el uso de un método top-down, dirigido por el negocio.

En RQ los requerimientos deben ser elicitados, definidos y documentados con una amplia perspectiva derivada de varios stakeholders y cubren todas las facetas de implementación en los desafíos del ciclo de vida.

Utiliza la descomposición de arquitectura usando una solución dirigida por casos de uso de negocio para construir el framework de servicios y técnicas análisis de reducción donde cada caso de uso puede dividirse en consumidor y servicios.

Los artefactos que maneja son:

- Entendimiento claro del sistema, servicios, prácticas y procesos que pueden ser mejorados.
- SOA Readiness Report, que es un reporte de las oportunidades relacionadas a los servicios los negocios más significantes, reuso, posibles brechas, factores críticos de éxito y riesgos.
- Recomendaciones tácticas y estratégicas y líneas guía (guidelines) para la adopción a SOA.
- Buenas prácticas, patrones de diseño, esquemas y recomendaciones.

Podría considerarse como puntos a favor las siguientes características:

- *Impulsa 17 características base para acelerar la adopción a SOA.*
- *Toma buenas prácticas de metodologías como RUP y XP y utiliza UML como herramienta de modelado.*
- *Impulsa el incremento en el nivel de comunicación a través de talleres tanto para personal técnico como para funcionales y del negocio.*

Sin embargo, una de las limitaciones que posee es que no provee suficiente información disponible y de acceso libre para analizar y profundizar de antemano sus características.

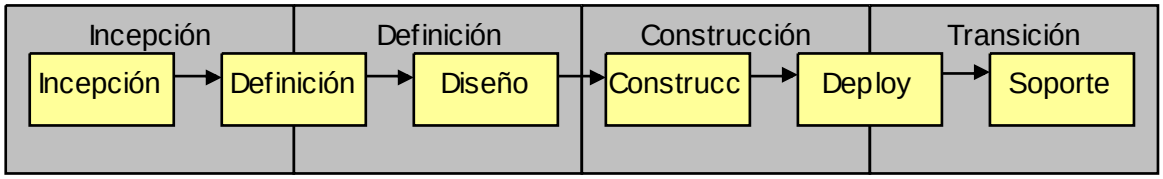
5.3.6 SOUP.

Sus siglas provienen de Service Oriented Unified Process. Es una metodología desarrollada por Kunal Mittal[96] que se basa en las buenas prácticas de RUP y XP con el fin de construir y luego optimizar una Arquitectura Orientada a Servicios.

Su ciclo de vida comprende seis fases: Incepción, Definición, Diseño, Construcción, Deploy y Soporte.

Está dividida en dos categorías:

1) **SOUP para deployments iniciales de SOA**.



Incepción:

Se establecen las necesidades para el proyecto SOA. Se explican las bases SOA a los equipos del proyecto y se planifica una estrategia de comunicación.

Los artefactos en esta fase son:

- *Visión y alcance de SOA*: indica la visión global de SOA estableciendo el alcance del proyecto.
- *Estrategia SOA*: plan de alto nivel que explica cómo será llevado a cabo el proyecto.
- *Análisis ROI*: indicando costos y ganancias que brindará el proyecto.
- *Plan de comunicación*: de los equipos con equipos de otros proyectos y usuarios. Este asegura que los clientes comprendan y estén involucrados en el proceso.

Quienes definen las estrategias SOA deben aportar su experiencia en el dominio y ayudar a identificar los problemas del negocio y las formas y medios para dar forma a las operaciones.

Los analistas y administradores de proyecto deben analizar las ventajas de una solución SOA, teniendo en cuenta las operaciones internas de los clientes, interacciones con socios, proveedores, clientes y el modelo de negocio en general. Deben hacer también un análisis ROI de la estrategia SOA que recomendarán al cliente mostrando costos y beneficios a corto, mediano y largo plazo.

[96] http://www.kunalmittal.com/html/index.shtml

Definición:

Es una fase crítica donde los equipos involucrados deben participar en la definición de los requerimientos y casos de uso desarrollados como parte inicial de SOA. Involucra las siguientes actividades: recolección y análisis de requerimientos funcionales y no funcionales, creación de un plan de proyecto con tiempos y estimaciones, definición de la infraestructura técnica, definición y realización de casos de uso, documentación y definición de la arquitectura general.

Involucra los siguientes artefactos:

- *Documento de requerimientos funcionales* con la descripción de los procesos de negocio que luego serán servicios de negocio.
- *Documento de requerimientos no funcionales*: describe consideraciones de rendimiento, SLA (Service-Level Agreement), requerimientos de infraestructura, etc.
- *Casos de uso y realizaciones*: casos de uso para los servicios de negocio que se construirán.
- *Documento de arquitectura SOA*: incluyendo hardware y software.
- *Documento de aplicabilidad de SOA*: que explica qué proyecto entra en el alcance del proyecto SOA y cómo los proyectos en curso pueden incorporarse a SOA.
- *Documento de definición de infraestructura:* incluye la diagramas de infraestructura detallada, servidores en línea, ubicaciones y conexiones entre los servidores necesarios para implementar SOA.
- *Plan de proyecto:* incluye los hitos y dependencias.
- *Modelo de gobierno y soporte:* contiene una descripción de cómo SOA será soportada y gobernada incluyendo consideraciones como monitoreo de SLA y management.

SOUP recomienda el uso de un proceso de manejo de requerimientos formal y debido a esto recomienda RUP en lugar de XP (si bien XP se vale de las historias de usuarios). Considera que el artefacto fundamental aquí es el modelo de soporte y gobierno de formal tal que éste indique cómo la organización da soporte a SOA. Se define la infraestructura técnica requerida para soportar SOA, incluyendo estimaciones sobre servidores, infraestructura de red, etc.

Diseño:

Se confeccionan los artefactos de diseño en base a la fase anterior. Los arquitectos SOA deben asegurar que el diseño SOA presentado funcionará para proyectos específicos efectuando previamente una prueba de concepto.

Los artefactos que se deberían entregar son:

- Documento detallado de diseño que indica cómo son diseñados y construidos los servicios.
- Modelo de programación de la aplicación que incluye detalle de deployment, teniendo en cuenta tecnologías, estándares de codificación, procedimientos de deploy, etc.
- Modelo de base de datos.
- Plan de pruebas y aseguramiento de calidad.

Incepción:

Se tratan actividades de deploy e integración. Todo el proyecto es manejado por un plan de proyecto SOA pero a su vez, existen sub-equipos que trabajarán en actividades diversas de construcción. Involucra actividades del tipo: desarrollo iterativo, QA y pruebas iterativas, documentación de usuario, etc. Los entregables son: el código fuente, los resultados de las pruebas y documentación de código, actualizaciones y diseño.

Deploy:

Se hace el deploy de la aplicación a un ambiente productivo. Se debe contar con un plan que indique cómo otros proyectos ya existentes interactuarán con la nueva aplicación.

Los artefactos a entregar son: Modelo de deployment, Modelo de uso, y modelo de niveles de soporte.

Soporte:

En esta fase se asegura el soporte de SOA a los proyectos existentes sobre corrección de fallas y nuevas funcionalidades mediante actividades como ser: mantenimiento, entrenamiento, correcciones, etc.

2) **SOUP para proyectos existentes con SOA**.

Para este tipo de proyectos SOUP utiliza XP para definir sus líneas guía. A continuación se mencionan las fases involucradas.

Incepción:

Una vez que se tiene un proyecto con SOA en marcha, se verá la forma en que determinados servicios puedan consumir los servicios existentes y publicar nuevos. En esta fase se establece la visión y el alcance general del proyecto. Los entregables son:

- Alcance y visión de un proyecto individual.
- Estrategia de proyecto y proceso para un proyecto particular explicando cómo aportará valor a SOA.

- Análisis ROI con los costos y beneficios a tener en cuenta en decisiones de implementación.

Definición:

Se identifican los servicios que ya se disponen y los que se necesitarán construir para cumplir los objetivos del proyecto.

Involucra actividades de: recolección de requerimientos, identificación de servicios, estimación de tiempos y desarrollo de casos de prueba.

Los artefactos a entregar son:

- Documento de requerimientos funcionales. El relevamiento no necesariamente debe ser llevado a cabo con estándares formales.
- Casos y de uso y sus realizaciones que cubran los nuevos procesos de negocio a construir.
- Documento de aplicabilidad de SOA indicando cómo el framework SOA existente se aplica para el proyecto.
- Plan de proyecto incluyendo hitos y dependencias.
- Estrategia de servicios que identifica servicios existentes y que pueden ser consumidos por los nuevos.
- Plan de pruebas que de acuerdo a XP son construidos en primera instancia antes de la codificación.

Diseño:

Solo se preocupa por cómo serán usados los nuevos servicios, que otras necesidades existen para su construcción y qué servicios hay que construir.

Los entregables al igual que antes, son: Documento de diseño detallado y Modelo de base de datos pero esta vez para un proyecto específico.

Construcción:

Incluye el ensamblado del nuevo desarrollo. Se supone que el desarrollo cada vez será menor debido al reuso. Se utilizan las técnicas XP para desarrollo iterativo con ciclos de hasta dos semanas. Incluye actividades de desarrollo y pruebas de forma iterativa más la documentación de usuario.

Los artefactos a entregar son: los nuevos servicios, resultados de las pruebas y la documentación.

Deploy: Se despliega una la aplicación que consume otros servicios o un nuevo servicio.

Soporte: del nuevo servicio.

Esta metodología, adopta las buenas prácticas y experiencias de la metodología RUP para los proyectos que se están iniciando en SOA y de XP y RUP para el mantenimiento del desarrollo de SOA.

Contempla el ciclo de vida completo del desarrollo de software estableciendo en cada fase los objetivos, artefactos a generar e hitos a cumplir.

A pesar de lo mencionado anteriormente, todavía no se encuentran aplicaciones en la industria como refuerzo a sus beneficios o limitaciones.

No se encuentra hasta el momento documentación específica que indique la forma de adaptarla [SOA-SURV].

5.3.7 Lean Software Development

Un estudio efectuado por IBM [LEAN-SOA] muestra el posible desarrollo de un proyecto con SOA mediante la metodología Lean Software Development en secciones anteriores descripta.

El objetivo del estudio fue determinar la forma en que los principios de Lean podrían beneficiar el desarrollo de un proyecto con SOA.

Como resultado del estudio se pudo ver que SOA y las prácticas ágiles tienen cuestiones en común. Por ejemplo el refactoring en SOA podría implicar modificaciones en las interfases y en la composición de servicios. Para soportar esto se requiere capacidad de adopción a los cambios y de la agilidad necesaria para su posterior implementación.

A continuación se incluye un estudio de cada principio de Lean sobre SOA también tomado del estudio de IBM antes mencionado.

1) *Eliminar desperdicios*
    a. El modelo de desarrollo de servicio **breadth-first** (a diferencia del depth-firts) permite diferenciar lo que es importante de lo que no lo es. Como práctica se recomienda evitar la generación de documentos de análisis up-front con demasiados detalles de partes insignificantes del sistema que nos pueden distraer de las que son críticas. Esto permite primero definir qué servicios se incluirán en el portfolio de servicios y para luego entrar en sus detalles. Existen técnicas en SOMA que permiten esto (descomposición del dominio top-down, análisis del sistema existente bottom-up y modelado de objetivos del servicio).

b. **Value Stream Mapping** permite determinar los desperdicios en procesos a través del estudio de productos y su valor frente a los clientes. Luego se pueden estudiar aquellos que aporten el mayor valor. Esta técnica también se utiliza para analizar los procesos de negocio que deberán ser soportados por SOA o el proceso de desarrollo de software para SOA .

2) Amplificar el aprendizaje

a. Dado que es casi imposible diseñar a priori una interfaz de servicio que sea consumible a futuro por todos los demás servicios, es necesario comenzar con una interfaz prototipo (que luego será expandida), hacer el deploy y luego obtener el **feedback** del resto de las aplicaciones. A partir de él se hará crecer el prototipo de forma tal que permita el reuso del servicio y que cumpla con todas las necesidades.

b. La **sincronización** es de especial importancia en la fase de implementación de un servicio donde a partir del análisis detallado de los servicios en esta fase, pueden ocurrir cambios en el alcance de los mismos.

3) *Tomar decisiones lo más tarde posible* en SOA significa mantener la interfaz del servicio abierta hasta que exista una evidencia clara de cómo ésta debería ser, en lugar de crearla sobre supuestos. De esta forma se requerirá de la sincronización con el resto de los servicios de la compañía de forma asidua en pos de un mejor modelo de servicio.

4) *Entregar lo más a menudo posible*

a. Pull Systems en el desarrollo ágil permite a las personas asignarse por sus propios medios las tareas a realizar mediante por ejemplo, pizzarras donde figuran todas las historias de usuarios en cartas. La asignación del trabajo consta de mover una carta del área de “tareas a realizar” al área de “tareas tomadas”. Para el diseño SOA se puede reemplazar el uso de CRC (clase, responsabilidad, colaboración) del diseño orientado a objetos por cartas SRC.

b. Modelo económico de contrato (Economic Model of Engagement): Se puede beneficiar un contrato SOA si se crea un modelo económico simple y se usa para tomar las decisiones de desarrollo. Con él, los miembros de cada equipo puede determinar lo que es importante del negocio trabajando todos sobre las mismas bases.

5) *Potenciar el equipo*: teniendo en cuenta que las personas más aptas para tomar las decisiones son aquellas que se encuentran cercanas a la realización del trabajo, permitiéndoles tomar algunas decisiones se puede asegurar que el código resultante para el servicio satisfaga la funcionalidad requerida. En SOA esto es beneficioso,

dado que se permiten delegar decisiones de diseño a los desarrolladores del servicios, logrando la toma de decisiones de forma colaborativa y participativa.

6) *Mantener la integridad*: en SOA la forma de mantener la integración es mediante comunicaciones fluidas entre el cliente y el equipo de desarrollo y entre cada subida y bajada del equipo de desarrollo. Si la comunicación se mantiene se asegura que las entregas cubrirán los requerimientos actuales y a futuro de los procesos de negocio.
    a. **Refactoring**: Con la alineación del negocio y la necesidad de asegurar que el servicio entregado se adhiera a una ambiente de negocio cambiante, existe una necesidad inherente de asegurar que el diseño del servicio subyacente es revisado y modificado continuamente. Una falla en esto podría ocasionar un servicio poco flexible que no soporta los cambios en el negocio. El continuo refactoring en SOA puede ser controlado mediante el modelo de servicio.
7) *Optimizar el todo*: En SOA como en cualquier otro contrato de arquitectura empresarial existe una necesidad inherente de que se mantenga la planificación global y el diseño detallado de los servicios a ser desplegados. Si se cae en la tentación de optimizar partes individuales o servicios a expensas del todo, se corre el riesgo de entregar una arquitectura inflexible que no esté alineado a las claves de los procesos de negocio dentro de la empresa.

Ver sección 3.2.8.5 donde se incluyen las ventajas y ventajas acerca de la aplicación de ésta metodología.

5.3.8 CBDI-SAE.

CBDI-SAE (CBDI-Service Architecture & Engineering) de Everware-CBDI Inc. es una estrategia para entrega y manejo de las actividades en el ciclo de vida de SOA. Surgió a partir de la documentación de experiencias SOA en sus foros[97]. Forma parte de CBDI-SAE SOA Referente Framework.

CBDI-SAE define principalmente una arquitectura de referencia y procesos. Se propone soportar estándares de desarrollo y sugerir que las empresas vendedoras de soluciones aplicables en SOA implementen sus herramientas de acuerdo a estándares basados en meta modelos para que realizar el trabajo sea más fácil y efectivo. Además

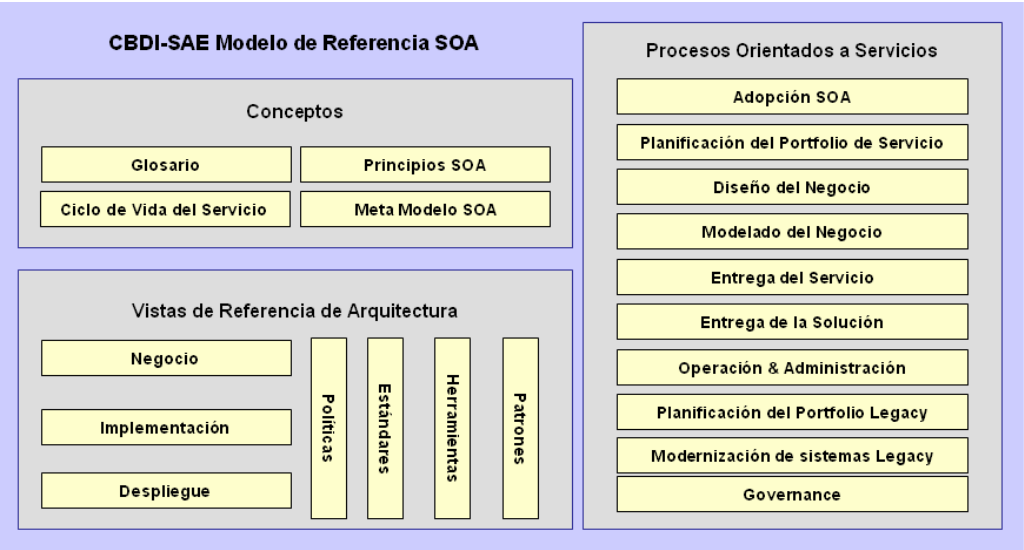
[97] http://www.cbdiforum.com

provee un modelo de referencia que comprende conceptos definidos, una arquitectura de referencia y procesos para que ellas posean un conjunto consistente de conceptos, políticas de inicio y estándares en un proceso repetible [CBDI-REP].

Modelo de referencia CBDI-SAE

Es un framework que extiende el modelo de referencia de OASIS[98] con el objetivo de proveer una vista conceptual más detallada a un nivel profesional, que pueda simplemente ser mapeado al modelo OASIS, más una arquitectura de referencia y modelos de procesos que informen las tareas de arquitectura.

Las partes esenciales en este modelo de referencia son los procesos y las buenas prácticas, dado que proponen ser guía para la arquitectura, aprovisionamiento, ensamblado de la solución y tareas de tipo operacionales y de gobierno. También deja en claro su foco en estándares, protocolos, herramientas, patrones y la identificación de políticas y gobierno.



CBDI-SAE Modelo de Referencia SOA [CBDI-REP]

Los principios sobre los que se basa SBDI-SAE son:

- Nivel de empresa: atiende a las necesidades de la empresa en su totalidad.
- Dirigido por políticas: separación de la designación de políticas de las responsabilidades del proyecto.
- Basado en gobierno: asegurando que se cumplan las políticas.
- Estandarización: reducción de la complejidad a través de servicios compartidos para el negocio y la infraestructura.

[98] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

- Abstracción: permite la agilidad entregando servicios abstractos que puedan soportar varios contextos.
- Composición: soporta líneas de requerimientos del negocio específicas que reusan e incorporan estándares del negocio y servicios tecnológicos.
- Modularidad: que reduce la dependencia y por esto mismo el horizonte de cambio.
- Virtualización: crea transparencia sobre los recursos subyacentes y luego la independencia de proveer y demandar decisiones del ciclo de vida y adquisiciones.
- Dirigido a futuro (Future Driven) comprensión de las necesidades reales del negocio para la agilidad e estabilidad de ingeniería donde sea apropiado y donde sea realmente necesario.
- Arquitectura inherentemente ágil: implementa una arquitectura de capas que facilita el cambio.
- Basado en modelo: usa modelos de forma proactiva para manejar el alcance a los principios acordados.

Meta Modelo SOA

Un meta modelo define las reglas de construcción de un modelo de negocio o software. Son importantes dado que nos obligan a definir los conceptos SOA con los que se trabaja, a eliminar redundancias entre conceptos, a definir atributos y propiedades de cada concepto, sus relaciones (y multiplicidad), etc.

Este meta modelo ayuda a mejorar la precisión de las técnicas y procesos en CBDI-SAE. Está documentado como modelos de clases UML compuesto de casi 60 clases, sus relaciones y atributos.

Arquitectura de Referencia

Define una arquitectura de referencia partiendo de un framework que unifique de forma cohesiva las áreas y componentes de una arquitectura empresarial convencional con los cambios que introduce SOA.

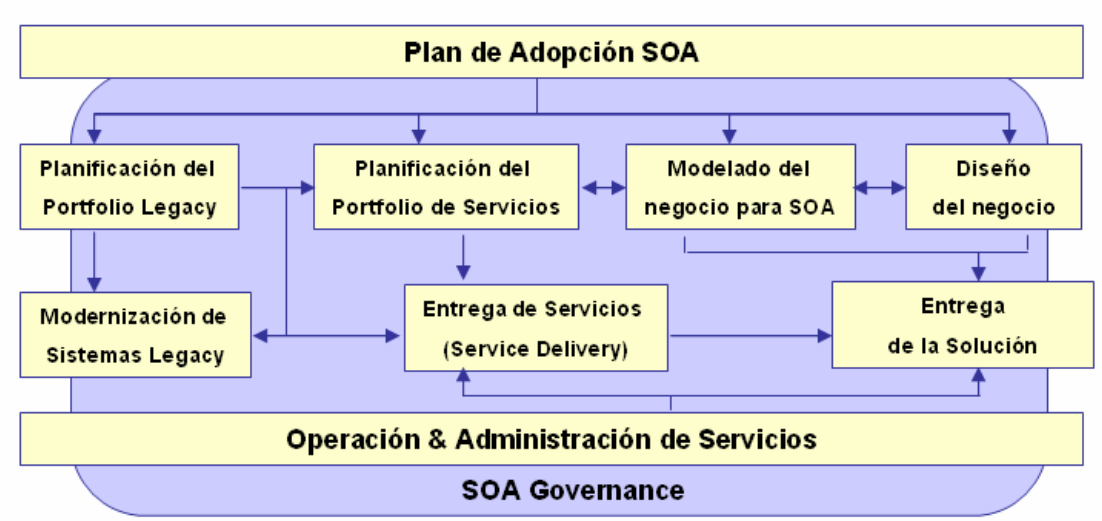
Comprende una serie de áreas que proveen una estructura de clasificación para documentar librerías o base de datos que contengan o hagan referencia a estándares, herramientas, políticas, plantillas, patrones y componentes reusables que formen la arquitectura de referencia SOA.

Procesos orientados a servicios

Se plantea la necesidad de que los procesos de IT evolucionen de forma tal que establezcan un framework de arquitectura y planificación más riguroso que coordine el desarrollo de servicios comunes compartidos y permita una solución de entrega de

proyectos que use servicios compartidos mientras conserva el grado apropiado de libertad siendo susceptible a los cambios.

Se incluye un diagrama con los bloques constructivos más importantes de una arquitectura de servicios e ingeniería de procesos.



SAE - Framework de Procesos [CBDI-REP]

Los procesos SOA deben ser distribuidos y estructurados y repetibles. Asimismo plantea los componentes que comprenden los nuevos procesos IT.

- Plan de Adopción SOA: un framework (Adoption Roadmap framework) define la capacidad y madurez requerida para entregar y manejar los aspectos de ambientes basados en SOA sobre una empresa altamente distribuida proveyendo las bases para la planificación, manejo, medida y monitoreo de las capacidades de la infraestructura organizacional y tecnológica.
- Planificación del Porfolio de Servicios (SPP): es un framework que identifica y divide servicios y capacidades de acuerdo a las políticas.
- Delivery de servicios: el proceso de provisión de servicios define una solución rigurosa para la especificación de servicios concluyendo en una alta calidad de implementación. Procuración, pruebas y certificación de procesos.
- Delivery de la Solución: un framework para ensamblado y personalización de servicios provee patrones y técnicas para la personalización y especialización de servicios comunes compartidos para conocer diversas necesidades de varias unidades de negocio mientras maximiza los niveles apropiados de datos del negocio y consistencia de reglas.
- Operación & Administración: un framework para la entrega y manejo de la infraestructura operacional soportando y manejando la ejecución de servicios incluyendo patrones de seguridad.

- Delivery de servicios: procesos e infraestructura implementando el ciclo de vida completo del servicio manejando los procesos IT modificando servicios a través de cambios de estado desde Planificado a Retirado incluyendo los requerimientos de governance.
- Modelado & Diseño del negocio.
- Planificación del Portfolio de Sistemas Legacy: un proceso sistemático para establecer el vínculo entre la administración de las aplicaciones legacy y el soporte para SOA, creando mapeos entre los artefactos legacy y el Plan de Portfolio de Servicios.
- Modernización de Sistemas Legacy: procesos definidos para el desarrollo, transformación y modernización de aplicaciones legacy a un ambiente SOA.
- Governance: proceso completo del ciclo de vida que cumpla las políticas SOA.

Un punto a favor, es que especifica un plan de adopción SOA. Además, como framework, se basa en estándares, protocolos, herramientas y patrones y en la identificación de políticas y gobierno.

Sin embargo, hasta el momento no se encuentran implementaciones llevadas a cabo mediante este framework que permitan determinar mejor, sus beneficios y limitaciones en la práctica.

5.3.9 Service-oriented analysis and design (SOAD).

SOAD ó Análisis y Diseño Orientado a Servicios, fue creado por IBM con el fin de establecer una metodología para el modelado y desarrollo en arquitecturas orientadas a servicios; partiendo de la base de que el modelado es una técnica poderosa para el desarrollo de soluciones SOA y que los procesos de desarrollo como el análisis y diseño orientado a objetos (OOAD), frameworks Enterprise Architecture (EA) y Business Process Modeling (BPM) solo cubrían una parte de los requerimientos necesarios para soportar los nuevos patrones de arquitectura en SOA. A ésta metodología también se la llamó “service-oriented modeling” (modelado orientado a servicios) [SOA-ELEM].

Incorpora innovaciones en lo que respecta a repositorios de servicios, orquestación de servicios, Enterprise Service Bus (ESB) y permite diseñar, construir agregar y hacer deploy de aplicaciones basadas en tecnologías SOAP, WSDL y UDDI[99].

A continuación se describen brevemente las limitaciones mencionadas por IBM[100] que motivaron el desarrollo de SOAD.

[99] http://en.wikipedia.org/wiki/Service-oriented_analysis_and_design
[100] http://www.ibm.com/developerworks/webservices/library/ws-soad1/

- Los *frameworks actuales de EA* no proveen asistencia total a los arquitectos SOA en la definición de la vista global de nivel de negocio. Por otra parte, generalmente hacen referencia a framew orks genéricos que no brindan soluciones concretas ni consejos tácticos a arquitectos y desarrolladores.
- *BPM* podría ser impulsado como un punto de inicio para SOAD, sin embargo debería ser suplementado con técnicas para generar servicios candidatos y sus operaciones para los modelos de procesos. El modelado de procesos en SOAD debe ser sincronizado con el modelado de casos de uso de nivel de diseño, y responder a las preguntas relacionadas con BPEL que se vayan dando como por ejemplo: qué aspectos deberían ser descriptos en BPEL y cuáles en WSDL, cómo podrían incorporarse cuestiones del tipo requerimientos no funcionales y características de calidad de servicio a los modelos, cuáles son las mejores prácticas a aplicar para asegurar la calidad de un servicio, etc.
- Dado que el *análisis orientado a objetos* es sumamente poderoso y debido a que SOAD es predominantemente de procesos en lugar de dirigido por casos de uso, se requiere un vínculo fuerte entre BPM y las actividades de modelado de casos de uso.

La diferencia entre orientación a objetos (OO) y la orientación a servicios (OS) radica en que en la primera el nivel de granularidad está puesto en el nivel de clase, lo cual reside a muy bajo nivel de abstracción para el modelado de servicios. La OS intenta lograr la flexibilidad y agilidad a través del bajo acoplamiento y la OO asociaciones como es la herencia crean un alto acoplamiento (y su posterior dependencia) entre las partes involucradas. Sin embargo, si bien éstas cuestiones de la OO dificultan la alineación de la OS de forma inmediata la OO sigue siendo una solución importante en el diseño de clases subyacentes y de la estructura de componentes dentro de un servicio. Además, técnicas de OOAD como son las tarjetas de clases, responsabilidades y colaboraciones (CRC) se pueden emplear en el modelado de servicios si poseen un elevado a nivel de abstracción.

Los requerimientos que propone son:

- Definición formal de procesos y notaciones como en otro proyecto.
- Conceptualización de los servicios de forma estructurada por ejemplo: uniendo clases y objetos a nivel aplicación con modelos de procesos dirigidos por eventos de BPM, empleando en primer lugar soluciones con orientación a eventos y procesos de negocio y en segundo a casos e incluyendo sintaxis, semántica y políticas.
- Recomendación de factores de calidad y buenas prácticas.
- Análisis de servicios para determinar cuáles de ellos no son correctos.
- Modelado extremo-a-extremo con una herramienta que lo soporte.

SOAD define factores de calidad como líneas base de diseño:

- Los servicios bien construidos facilitan la reconfiguración y el reuso a través del bajo acoplamiento, encapsulamiento y ocultamiento de la información.
- Los servicios bien diseñados son aplicables para más de una aplicación, las dependencias entre servicios son minimizadas y explícitamente indicadas.
- Las abstracciones de servicios son cohesivas, completas y consistentes.
- Los servicios son generalmente stateless.
- El nombramiento de los servicios es entendible para los expertos del dominio que no poseen experiencia técnica.
- Los servicios siguen la misma filosofía (con patrones y plantillas) y patrones de interacción; el estilo de arquitectura subyacente es fácilmente identificable.
- El despliegue de los servicios solo requiere un perfil de programación básico junto con el conocimiento del negocio; cuestiones de middleware solo concierne a algunos pocos especialistas.

Propone la identificación de servicios por medio de:

- Técnicas de análisis directo del negocio (por ejemplo las entrevistas con clientes) reforzadas por métodos indirectos. Entre otras cosas sugiere entrevistar a quienes se encargan del manejo del producto y a líderes del negocio, consultar los diagramas organizacionales sobre el que se construirá el sistema, consultar casos de uso existentes, utilizar la terminología que se emplean en las presentaciones de marketing.
- Descomposición del dominio, análisis de subsistemas, creación de un modelo de objetivos, etc.
- Modelar con la mayor granularidad posible sin perder relevancia, consistencia y completitud.
- Definir convenciones de nombres ya sea para namespaces, paquetes java, dominios de Internet, etc.

A su vez destaca cuestiones a profundizar:

- Categorización y agregación de servicios de acuerdo a su propósito y uso. Por ejemplo los servicios atómicos deben ser orquestados en un nivel más alto.
- Políticas y aspectos. Se requieren frameworks de políticas para servicios web debido a que los WSDL no siempre describen todas las características de un servicio. Además se requiere tener trazabilidad del negocio para poder vincular en cualquier momento todos los

artefactos en runtime mediante algún lenguaje que pueda ser entendible con alguien sin perfil técnico.

- Método meet-in-the-middle para poder integrar a los sistemas legacy.
- Definir la semántica de los parámetros de los contratos interfases de servicios.
- Verificar que los servicios sean reusables.

Como punto a favor se puede indicar que se basa en las buenas prácticas del análisis y diseño orientado a objetos y BPM. Impulsa el uso de estándares como ser WSDL, UDDI y XML. Además, incorpora la conceptualización, categorización y agregación de servicios, políticas y aspectos, proceso meet-in-the-middle y recolección de servicios [SOA-SURV].

Sin embargo, solo contempla el análisis y diseño debido a que es un framework abstracto en lugar de una metodología que contemple el todo [SOA-SURV].

5.3.10 Steve Jones' Service Architectures.

Su objetivo en cambio es proveer un mecanismo para la planificación, administración y delivery de proyectos usando técnicas SOA. Considera que mientras que otras metodologías (como por ejemplo RUP y XP) se enfocan estrategias de delivery y en los artefactos dentro del proyecto, ésta consiste en un proceso de descubrimiento top-down centrado en el "qué", es decir de qué se trata el negocio en lugar de centrarse en el "cómo" (la implementación). Para ello adopta un proceso de cuatro pasos donde se analiza, el qué, quién, por qué y cómo. El qué define el alcance del servicio, el quién se enfoca en los actores externos con los que el servicio interactúa, el por qué se centra en el por qué un servicio se comunica con otro y por qué los actores interactúan con él, y por último, el cómo detalla los procesos que coordinan los servicios y detalles de cómo podría asimismo implementarse el servicio. Una arquitectura de servicios define el qué, identifica el quién, subraya el por qué y no dice acerca del cómo [JONES-METH].

Las sugerencias que indica son:

- Crear una gran foto que oficiará de guía para el proyecto proveyendo una vista de cómo la organización o el proyecto divide sus capacidades en servicios.
- Comenzar desde la cima del dominio debido a que las organizaciones operan de forma top-down, se reduce el desorden y usa como base las funciones de la organización, es decir el qué y no una representación temporal como lo es un diagrama organizacional.
- No comenzar con procesos debido a que los procesos basados en descubrimiento tienden rápidamente a enfocarse en los detalles, a producir procesos silos de servicios y a

menudo fallan en la identificación de cuestiones comunes entre procesos. Debido a que las organizaciones se enfocan en primer nivel en el qué y luego en el cómo, recomienda no comenzar con procesos, dado que éstos son el cómo.

- La clave de una arquitectura de servicios es el proceso que va desde la toma de la información hasta la comprensión de la misma mediante el trabajo colaborativo que consiste en crear un diálogo común entre varios grupos y decidiendo los límites del negocio. En el inicio del proyecto se sugiere iniciar con sesiones intensivas de 1 a 3 días dependiendo de la información disponible y las decisiones a tomar. En ellas deben estar reunidos los stakeholders, las personas que trabajen dentro y fuera de la empresa y quienes entiendan del negocio.
- Una vez que se tiene la primera versión de la infraestructura de la arquitectura de servicios es necesario que se efectúen revisiones que se inicien como parte normal del negocio y los procedimientos de cambios tecnológicos, considerando previamente el impacto sobre la arquitectura de servicios. En caso de no tenerse esas revisiones, se recomienda llevar a cabo períodos de revisiones mínimas: nivel 0 una vez al año, y nivel 1 más de un año y cuarto. Las revisiones no deben intentar recrear la arquitectura sino enfocarse en los cambios que podrían impactar partes de la arquitectura.
    - Si se identifican grandes cambios en cuanto a la operatoria de la empresa, se debe efectuar una revisión de la arquitectura y potencialmente crear una nueva. En caso de una nueva arquitectura se deberían implementar en paralelo de forma tal que se pueda comparar la nueva con la que ya existía con el objetivo de identificar discrepancias y servicios comunes.
    - Para determinar la arquitectura de servicios se recomienda tener personas imaginando que ven a la empresa, proyecto o departamento desde afuera. Luego hacerles preguntas de alto nivel de forma tal que permitan conocer los servicios sin entrar en mucho detalle.

Steve Jones indica la forma de crear una arquitectura de servicios a partir de dos escenarios, uno describiendo la arquitectura de servicios completa para una organización y la segunda para un proyecto específico. Como primer hito se menciona la identificación de servicio, la forma de crear una arquitectura de servicios, de desarrollarla y por último administrar los cambios.

Identificación de servicios:

1) Plantilla de definición: incluye el tipo de información que debería relevarse en la definición de un servicio. Identificando las prioridades de los stakeholders clave y

actores se pueden identificar los elementos importantes y las tareas potenciales a realizar. Se puede construir una planilla que incluya: nombre del entrevistado, rol que desempeña, descripción de su cargo y responsabilidades y por último prioridad. Por otra parte otra planilla que identifique las tareas principales, relevando los siguientes datos: tarea, descripción, pre y post condición y por último las cosas que no cambian luego de efectuada la tarea.

2) Nivel cero: solo se consideran los servicios centrales del negocio. Cada sistema de nivel cero podría considerarse como un área aparte y su potencial reemplazo podría provocar un impacto en los otros servicios del dominio. Otra forma de saber qué servicios son de nivel cero es teniendo en cuenta que si se combina el servicio de nivel 0 en un dominio mayor, no se reduciría la claridad del sistema. Generalmente puede llegar a haber de 1 a 5 servicios en el nivel 0.
    a. Modelo de empresa de nivel cero: para identificar los servicios hay que saber cuáles son las áreas clave del negocio que componen la empresa y luego los actores clave que interactúan externamente con el servicio y no aquellos internos que proveen el servicio.
    b. Modelo de proyecto de nivel cero cuyo objetivo es comprender el objetivo del proyecto y el dominio que cubre. Para comenzar se tiene que saber qué se requiere del proyecto en si. Se pueden crear diagramas de actividades como representaciones de diferentes escenarios u opciones.
3) Pasaje al Nivel 1: se hace mediante la descomposición de los servicios de nivel o. Los servicios de éste nivel pueden identificarse mediante las áreas donde la gente trabaja día a día y los servicios IT implementados para soportarlos. Se deben tener en cuenta dos cosas para la descomposición de servicios. La primera se refiere a que el servicio que los engloba brinda comportamiento y administración a los servicios de menor nivel. La segunda cuestión es que el propósito de clasificación es permitir la navegación simple de la arquitectura de servicios para asegurar que futuros servicios puedan ser clasificados fácilmente dentro del modelo de servicios.
    a. Nivel 1 de empresa: Los servicios se pueden asociar con los departamentos del negocio. La importancia de un servicio es que representa el “qué” y los departamentos generalmente representan el “cómo”.
    b. Nivel 1 de proyecto: Los servicios se pueden asociar a los roles y áreas de aplicaciones IT requeridas. En el modelo se establecen las interacciones con el exterior y las principales dentro del modelo de servicio. Puede usarse como guía un máximo de 8 niveles 1 por cada nivel 0 aunque generalmente son 4.

4)

a. El refinamiento posterior se puede realizar con dos objetivos. El primero consiste en hilar en profundidad y entender más el dominio problema mediante un proceso similar al del nivel 0 y 1 obteniéndose descomposiciones en servicios de nivel 2. El otro es para un servicio dado enfocarse en diferentes representaciones externas que podría tener y se usa para servicios que interactúan con un conjunto de funcionalidades comunes de diferentes formas.
b. Los servicios virtuales se usan cuando servicios internos son combinados para proveer una vista externa para los clientes. Éstos ofrecen una fachada sobre el resto de los servicios. Pueden usarse cuando un servicio provee objetivos operacionales diferentes dependiendo de cómo es invocado. Proveen una forma de indicar donde será coordinada y potencialmente simplificada la lógica del negocio, sin embargo la implementación de la lógica actual solo será realizada en otros servicios no virtuales.
c. Los servicios de soporte son elementos tecnológicos que al negocio no le interesa que existan. Se dividen en dos grupos: los técnicos (que proveen soporte a las funciones del negocio en lugar de ser funciones de negocio específicas en si, por ejemplo servicios de impresión, OCR, etc.) y los asociados (no son requeridos para las operaciones del negocio del sistema pero son requeridos para que el negocio pueda operar, por ejemplo servicios de recursos humanos).
d. Los servicios compartidos son comunes a múltiples áreas de negocio, pueden ser técnicos o de soporte.

Creación de la arquitectura de servicios:

1) Durante la etapa de pre-trabajo se debe lograr un nivel de conocimiento suficiente, para luego poder crear la arquitectura de servicios. Se surgiere comenzar con un glosario de forma tal que todos entiendan lo mismo para cada término. Se deben identificar los stakeholders requeridos para crear la arquitectura de servicios de nivel 0 y 1 y planificar eventos. Durante esta fase se crean las herramientas colaborativas iniciales para compartir los resultados de cada trabajo. Los entregables sugeridos para esta fase se muestran a continuación.

Entregable	Descripción	Duración de empresa	Duración del proyecto
Sponsorship	Sin un claro apoyo ejecutivo los esfuerzos de construcción fallan.	No aplica. Si no se logró entonces la etapa 0 se alcanzó.	No aplica. Si no se logró entonces la etapa 0 se alcanzó.
Identificación de stakeholder	Del negocio, tecnología o de interacciones externas para la participación en el evento.	De 3 días a 2 semanas según escala y ambiente.	Generalmente 3 días.
Planificación de evento.	Planificación y definición del alcance de evento.	De 2 a 6 semanas según escala y cantidad de participantes.	Entre 1 y 3 semanas según escala de proyecto y logísticas involucradas para reunirlos a todos.
Soporte técnico	Herramientas colaborativas y disponibilidad de información como soporte del evento.	1 semana de ajustes.	1 semana de ajustes.
Recolección de información	Información específica para el evento.	De 2 a 6 semanas según la profundidad y cantidad requerida.	1 a 3 semanas según la profundidad y cantidad requerida.
Definición del contexto	Definición de la misión para el evento.	2 días con el sponsor del evento.	1 día con el sponsor del evento.

2) El evento tiene el objetivo de reunir a los stakeholders de acuerdo en sus posiciones, debe estar bien planificado con los horarios de inicio y fin de cada tarea, provisto de máquinas con acceso a Internet y se deben usar pizarras.
    a. El *facilitador* debe tener habilidades en el manejo de comunicaciones, poder de escucha, comprensión del dominio, coordinación de equipos, tener la capacidad de finalizar conversaciones con un acuerdo final y poseer experiencia previa.
    b. El *cuadro de nivel 0* permite determinar cuáles son los servicios actuales sin preocuparse acerca de sus interacciones. El objetivo es identificar los servicios de nivel 0.
    c. *Agregar actores* externos que interactúen con los servicios.
    d. *Incorporación de interacciones* es decir, por qué varios servicios y actores interactúan. El objetivo es definir el contrato de valor para el servicio que ofrece al mundo.
    e. *El detalle de los servicios* se efectuará mediante la creación de grupos de personas que describan los servicios, cada grupo describirá los servicios que más conocen.
    f. *Entregables de nivel 0*.

Entregable	Descripción	Duración de empresa	Duración de proyecto
Diagrama de servicio	Diagrama de servicio de éste nivel	½ día.	2 horas.
Actores	Actores externos.	2 horas.	1 hora.
Interacciones servicio-actores.	Interacciones entre servicios y actores.	2 horas.	1 hora.
Definición del servicio.	Definición del servicio y sus interacciones en detalle.	4 horas.	2 horas.
Reporte	Confirmación de todo lo relevado.	30 minutos.	15 minutos.

g. *Pasar del nivel 0 a elementos de un menor nivel (Drilling-Down)* mediante la repetición de pasos del b al e creando los entregables que se muestran en la tabla anterior.
h. El nivel de *detalle del diagrama* que se vaya construyendo será guiado por el facilitador teniendo en cuenta que a mayor detalle menos claro será el diagrama.
i. *Publicación del cuadro* una vez finalizado.
j. *Fin del evento*. Se deberán definir tiempos y futuras acciones. Para un modelo de empresa, quizás la siguiente clave sería identificar los procesos de negocio de alto nivel que funcionen entre los servicios y para un proyecto identificar la tecnología actual involucrada en algunos servicios. Es importante que no pasen más de dos días desde terminado el evento para que cada integrante posea una copia de todo lo producido y por otra parte que el sitio donde se publican las cosas esté constantemente actualizado.

3) Algunas aclaraciones finales. No se debe tomar como servicio el "actualizar cliente" o "obtener cliente" dado que éstos son solo puntos de entrada para lo que sí es el servicio "cliente" dado que los servicios son colecciones de procesos.

Desarrollo de la arquitectura completa:

El modelo de servicio define el contexto, conceptos, objetivos para toda la arquitectura y dominios diferentes. Una vez creada la arquitectura se pone el foco, dentro del proyecto, en comprender dónde viven los servicios en la infraestructura del negocio y cómo son entregados a los usuarios. Dentro de una arquitectura empresarial es el proceso de comprensión de los servicios, IT, estrategias de vendedores, definición de los mejores estándares técnicos y de arquitectura y por último las prácticas para la migración la organización a una arquitectura SOA. Luego la arquitectura puede agregar a los servicios elementos no funcionales (restricciones de seguridad, rendimiento, redundancia y confianza, etc).

Administrar cambios:

La gestión de cambios es más sencilla en arquitecturas de servicios y debido a que los servicios se mapean con el negocio los cambios tienden a estar limitados dentro de esos límites. Por otra parte los servicios pueden ser mantenidos y versionados rápidamente.

Uno de los puntos rescatables de ésta metodología es que se basa en los primeros pasos necesarios (el qué, el quién, el porqué y el cómo), en un proyecto para asegurar que se cumplan las propiedades SOA en la entrega final [SOA-SURV]. Por otra parte, especifica con detalle, los entregables necesarios para cada fase.

Sin embargo, aún no se han encontrado ejemplos de aplicación donde se indiquen buenas y malas experiencias logradas.

5.3.11 BEA

BEA considera que SOA es una estrategia a largo plazo que exige mantener un enfoque consistente en transformar la manera de hacer funcionar la TI y responder a las necesidades empresariales de forma inmediata. Para ello indica que se debe mantener un equilibrio entre los objetivos a largo plazo y las necesidades de plazo menor mediante un conjunto de prácticas organizativas, financieras, operativas, de diseño y de distribución desde el inicio de SOA.

BEA establece un Modelo de seis Dominios con el fin de responder a los retos para presentar con éxito SOA. A continuación se incluye un gráfico que lo muestra y luego se da un breve detalle de cada uno tomando como referencia [BEA-DMOD].

Cada dominio si bien es diferente al resto, es interrelacionado con otros e interdependiente.

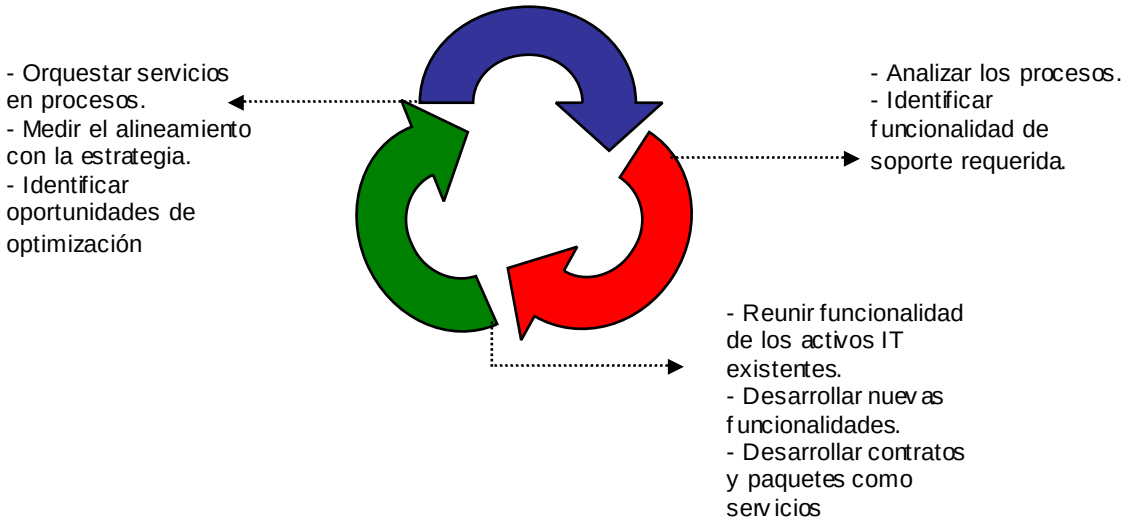
Modelo de Ámbitos de Dominio© de BEA

Estrategia y proceso empresarial: Las compañías necesitan implementaciones tecnológicas que soporten los negocios y sus necesidades cambiantes. Provee un ambiente que vincula la gestión y las mediciones de IT con la estrategia empresarial de forma que trabajen conjuntamente para lograr una mejora continua de los procesos.

Debido a que generalmente las compañías desarrollan una estrategia empresarial y luego intentan implementarla por medio de una estrategia de IT independiente y debido a que cada una de estas estrategias opera en escalas temporales diferentes (empresariales largo plazo y IT a corto), se obtiene un programa IT en silos que no da soporte a la empresa en su totalidad. El objetivo de BEA es lograr mediante un programa SOA el alineamiento de la estrategia IT con la empresa.

El programa SOA fomentará con este fin, entre otras cosas, controlar la estrategia SOA en toda la empresa, definirá una arquitectura dinámica con capacidad de respuesta basándose en estándares, garantizando resultados económicos favorables, decidirá las prioridades de desarrollo y despliegue de servicios eligiendo los incrementos y cuándo aplicarlos, fomentará el cambio y adopción mediante publicidad y establecimiento de incentivos, asegurará que se apliquen mediciones para proporcionar un análisis constante de costo-beneficio fomentando un ciclo continuo de feedback para cuestionar la viabilidad del programa y por último y no siendo menor organizará y controlará el seguimiento de los procesos, políticas y estándares.

El círculo continuo de feedback para el alineamiento con empresa permite la optimización de los procesos.



Arquitectura: propone un ambiente basado en estándares, distribución, bajo acoplamiento y representación de procesos de negocio diseñados para responder a los cambios e integrar funcionalidad a nivel empresarial.

La arquitectura deberá estar basada en:

- **Servicios**: marca un contraste con la forma tradicional de compartir funcionalidad a nivel de código o de componentes. En una arquitectura de servicios se piensa, factoriza y despliega una sola vez para su empleo en todos los niveles de la empresa ofreciendo ventajas relacionadas con la reducción de costos, rápida distribución y capacidad de respuesta de las IT ante los cambios.

- **Estándares**: a diferencia de las IT tradicionales donde cada proyecto exige el método más expeditivo para satisfacer los requisitos y en consecuencia de ello la proliferación de tecnologías que ocasionan dificultades en el intercambio de información (CORBA, DCOM), plantea el desarrollo con estándares XML, Servicios Web y UDDI como bases para SOA.

- Con **enfoque en la empresa**: mediante una arquitectura empresarial que brinde una plataforma de despliegue basada en servicios, correctamente construida y con principios de control adecuados. Los comités de arquitectura se deben centrar en: la selección de tecnologías, en el control, adoptando un mecanismo que defina el despliegue, supervice y gestione el acceso a la funcionalidad de la empresa de forma estándar con los niveles propicios de granularidad y visibilidad para los usuarios.

- Con **enfoque en el negocio**: mediante una arquitectura orientada a servicios que proporcione la funcionalidad de la empresa al nivel en que los usuarios entienden el negocio, haciendo más fácil al usuario comprender, especificar, probar y operar a diario.

- Empleando una **arquitectura de referencia** que describa los componentes principales y sus interrelaciones.

La *infraestructura de los servicios* permite la separación de los usuarios con de la funcionalidad empresarial de los sistemas y aplicaciones que proporcionan esa funcionalidad. La bases sobre las cuales se extienden los servicios son las aplicaciones, los datos y el middlew are. Los servicios de infraestructura incluyen:

- *Gestión de servicios* para proporcionar independencia de ubicación, tolerancia a fallos, gestión y otros atributos de calidad de servicio empresarial.
- Un *bus de servicios* para el enrutamiento común y capacidades de transformación de forma muy similar a la de un intermediario o bus de mensajes tradicional en soluciones middlew are comunes.
- *Servicios comunes* de conexión, auditoria, seguridad y manejo de errores. Se despliegan en una infraestructura compartida que es clave para construir con éxito una empresa orientada a servicios.

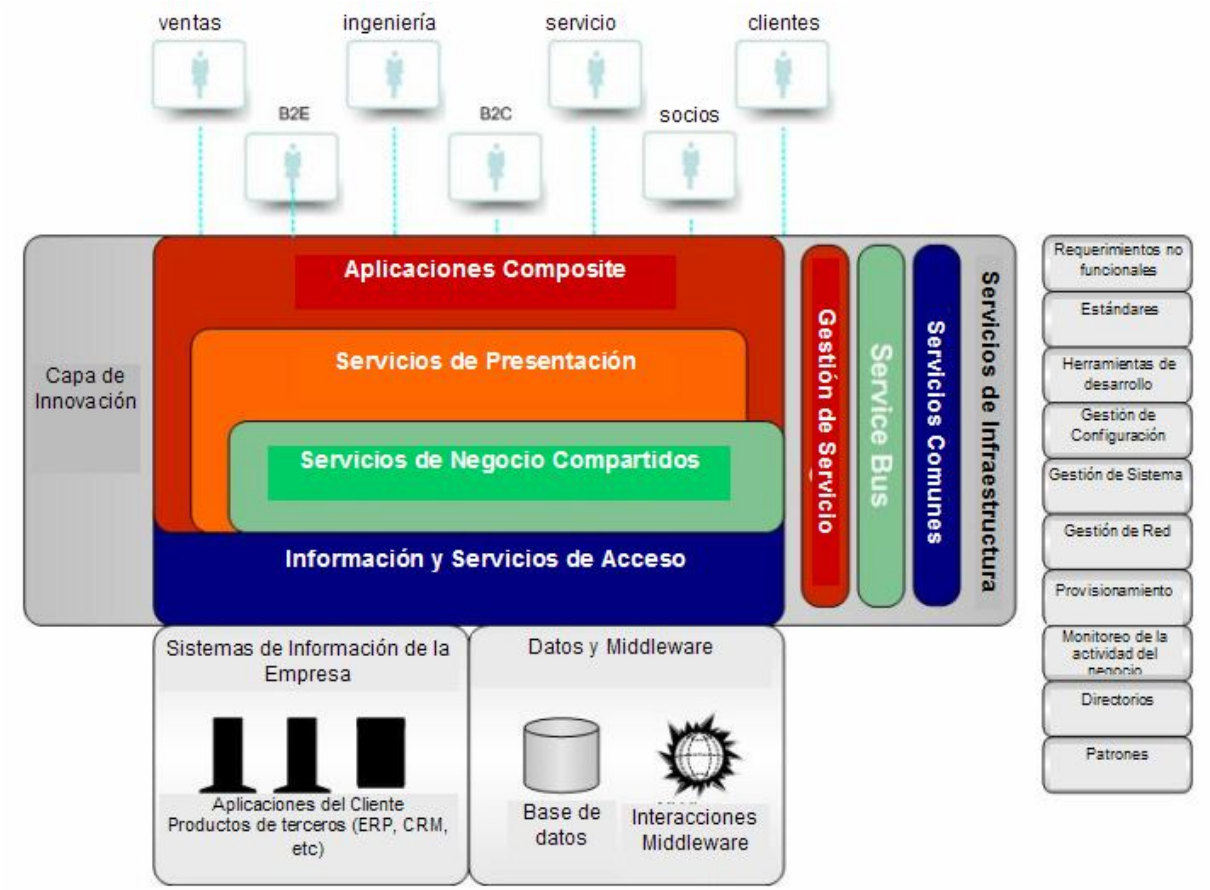
La *capa de servicios de acceso e información* representa la funcionalidad de la empresa existente y estandariza el acceso a la funcionalidad y a la información de la empresa de forma tal que los consumidores no necesiten conocimiento sobre la tecnología subyacente. Estos servicios se crean o se aprovechan a partir de recursos existentes en la empresa, y se despliegan sobre la infraestructura compartida para asegurar una calidad de servicio de categoría empresarial.

La *capa de servicios empresariales compartidos* representa la funcionalidad central de la empresa. Aprovecha los servicios de la capa de servicios de acceso e información para proporcionar funciones empresariales comunes.

La *capa de servicios de presentación* representa componentes comunes que emplean información empresarial compartida y servicios de acceso para interactuar con recursos empresariales. Esta capa proporciona presentación reutilizable.

La *capa de aplicaciones compuestas* orquesta otros servicios y funciones para distribuir funcionalidad de aplicaciones de mayor orden para la empresa. Esta capa representa funcionalidad empresarial en la forma en que los usuarios empresariales piensan y esperan usar tecnología.

A continuación se incluye el diagrama que representa las capas antes descriptas.



Bloques constructivos ó básicos: propone un cimiento común basado en estándares que permiten alcanzar la consistencia y maximizar la habilidad de repetir casos de éxito mediante el reuso de implementaciones y una infraestructura central.

Los bloques constructivos son los elementos reutilizables que se desarrollan y la infraestructura en que se despliegan, supervisan y gestionan. Se clasifican en bloques constructivos de software (código, modelo de datos, procesos, servicios, aplicaciones y componentes) y los bloques constructivos organizativos (buenas prácticas, estándares y herramientas para el desarrollo, despliegue, organización y gestión).

Se recomienda su construcción de forma incremental e iterativa para minimizar el riesgo y los costos.

Para la presentación de los primeros bloques constructivos se requerirá de una infraestructura que los soporte (por ejemplo un registro de publicación y descubrimiento servicios).

Cada bloque de construcción tiene su propio ciclo de vida de servicio:

- Identificación del servicio y diseño.

- Construcción del catálogo de servicios.
- Diseño del servicio.
- Construcción del servicio.
- Visibilidad del servicio.
- Deploy del servicio.
- Seguridad del servicio.
- Publicación y descubrimiento del servicio.
- Planificación del proyecto SOA.
- Governance.
- Versionado de servicios.
- Versionado de contratos.

Proyectos y Aplicaciones: cataloga, categoriza y modifica las funcionalidades ofrecidas por determinados sistemas para estandarizar la forma en la que esto se hace evitando la redundancia y promoviendo la consistencia en la ejecución del negocio.

Plantea la construcción de un mapa de servicios en forma de guía y establecimiento de prioridades para reutilizar, desarrollar y refinar servicios. Se inicia relevando las funcionalidades ya existentes para saber qué servicios implementar y cuáles reutilizar.

En una primera instancia fase del programa SOA debe crearse un *Inventario de Aplicaciones* (actuales) y un *Catálogo de Proyectos* (en marcha) que contenga información del tipo: funcionalidad, granularidad, interdependencias, costos y métricas en relación al desarrollo, modelos de datos, transformaciones y convenciones utilizadas, flujo de trabajo y de procesos implicados en aplicaciones, acuerdos de nivel de servicios (SLA), calidad de servicio (QOS) e información relacionada.

Una vez construido el catálogo se seleccionarán los servicios iniciales para la infraestructura SOA.

Un Catálogo de Servicios Empresariales publicará detalles de los servicios ya ofrecidos (interfaz, funcionalidad que proporciona, metadatos asociados en la definición de su contrato).

Organización y Control (governance en inglés): Asegurar las necesidades IT del negocio y maximizar el uso de funcionalidades desarrolladas mediante una estructura organizacional y mandato de gobierno.

BEA considera que la ausencia de principios de organización y control rigurosos y sostenibles es el factor principal de cualquier resultado insatisfactorio. Un programa sólido

de control de SOA controlará las siguientes áreas: cumplimiento de estándares, mapa de servicios (definición, desarrollo y despliegue), gestión de cambio empresarial, obligatoriedad de reutilización, estructura de organización (asegurar la adherencia a disciplinas como cumplimiento de estándares, reutilización, y el plan de trabajo del servicio), cambio de cultura, desarrollo de destrezas y buenas prácticas, modelo de financiación y responsabilidad.

El programa de control se desplegará en tres fases:

1) **Definición** de: Principios rectores de SOA, Arquitectura SOA, Directrices, Política y estándares, Procesos y procedimientos SOA, Organización SOA (diseño, roles, capacidades, responsabilidades).
2) **Gestión**: Comunicación (interna, externa), Estándares SOA y cumplimiento de servicios, Proceso de excepciones de Arquitectura SOA, Inspección de Arquitectura SOA y proceso de adaptación, Proceso de gestión de cambios.
3) **Soporte y control**: Tutelaje en toda la empresa, Definir y gestionar Catálogo de servicios empresariales, Supervisar procesos y Financiación.

El modelo de control abordará temas de control en términos de IT y empresariales. Con respecto a los primeros incluirá la definición de servicios compartidos y su reutilización, creación de servicios, consumidores y acceso a servicios, despliegue y operaciones de los servicios asociados, coordinación de actividades y principios para la gestión.

Por otra parte, en términos empresariales incluirá: la forma de medir la eficiencia en tiempo de puesta en mercado y rendimiento de la inversión y a qué nivel de granularidad para mantener la responsabilidad de la iniciativa SOA, hacer una análisis costo-beneficio, gestionar los grupos de servicios orquestados en soluciones empresariales, cómo ejecutar la ingeniería de dominios de plazo más largo para poder seguir optimizando los procesos empresariales representados en las IT mediante el repositorio de servicios.

Recomienda componer un comité SOA que responsable de auditar el nivel de disponibilidad de servicios y cumplimiento de contratos, resolver cuestiones presupuestarias, asegurar e imponer la reusabilidad, definir mecanismos de versionamiento y control de cambios, seguir métricas, costos, asegurar la adhesión a la arquitectura de referencia, etc.

Costos y Beneficios: Planear y ejecutar las implementaciones IT para crear valor de forma temprana y sustancial de forma tal que soporte las inversiones existentes sobre IT mientras se adapta al cambio y se crece.

5.3.12 Service Oriented Architecture Framework (SOAF).

SOAF es un Framework Orientado a Servicios desarrollado por IEEE.

Según [SOA-SURV] SOAF abarca cinco fases del proceso de desarrollo de software: elicitación de información, identificación de servicios, definición de servicios, realización de servicios, roadmap (mapa de ruta) y planificación.

Se basa en dos tipos de modelado: el modelado denominado “To-be” que posee una estrategia de delivery tipo top-down orientado al negocio que describe los procesos de negocio necesarios. Y el modelado “as-is” del tipo bottom-up que describe los procesos de negocio que se encuentran implementados tal como se encuentran en las aplicaciones existentes.

Una limitación que presenta es que no posee material disponible y de libre acceso, donde se explique en detalle éste framework.

Tampoco se han encontrado casos de estudio sobre desarrollos en SOA con éste framework.

5.4 Algunos ejemplos

En esta sección se incluyen algunas experiencias reales de implementación de metodologías de desarrollo sobre proyectos con base SOA. Los mismos permiten divisar beneficios y limitaciones obtenidas luego de su aplicación.

5.4.1 RUP, Agile, Project Driven y Shlaer-Mellor

John Harby quien es participante de varios comités técnicos de OASIS, en una entrevista efectuada el 24 de Julio de 2006 para InfoQ “SOA Integration and Methodologies”[101], indicó que de acuerdo a su experiencia, las metodologías más utilizadas eran las basadas en RUP, Agile y Project Driven, y la de Shlaer-Mellor.

[101] http://www.infoq.com/news/SOA-Methodologies

Explicó que RUP es recomendable para el diseño de servicios con bajo acoplamiento, pero está más orientada a los proyectos más que a la empresa en si. Por otra parte, la fase de identificación de servicios no está lo suficientemente madura. Además, se considera que quizás las metodologías basadas en proyectos podrían ser aplicables a proyectos SOA siempre y cuando sean pequeños proyectos, donde el proyecto tiene un ciclo de vida manejable y asimismo se orienta a las soluciones tradicionales de modelado dado que los ciclos de vida de todos los sistemas dependientes están alineados. Diferente sería en casos donde los proyectos SOA son más grandes donde los servicios de un sistema son consumidos por aplicaciones de contextos diferentes.

Las metodologías dirigidas por proyectos tienen el suficiente aporte en las fases iniciales pero luego la vida de SOA puede entremezclarse con la del proyecto. En esos casos notó que las necesidades del negocio para el proyecto desaparecían dejando los esfuerzos SOA débiles. Para estos casos se emplearon soluciones ágiles y se vio la efectividad sobre todo en áreas donde SOA estaba integrando funcionalidades existentes permitiendo que dos equipos de trabajo se unan. Para mantener actualizado el governance del proyecto se emplearon técnicas de Scrum.

Por último la técnica de Shlaer-Mellor [102]si bien es bastante compleja y anterior a RUP aportó en proyectos SOA donde participó a mediados de los noventa con C++/CORBA, patrones de diseño para entidades con bajo acoplamiento.

Ágiles

Debido a que tanto SOA como las metodologías ágiles (denominadas Agile en inglés) comparten el propósito de permitir la flexibilidad en el negocio y reducir el costo del cambio, ya sea mediante las prácticas ágiles o las buenas prácticas SOA mencionadas anteriormente, a continuación se incluyen varios planteos que sugieren la aplicación de métodos ágiles para proyectos que poseen SOA.

Digital Focus, una empresa especializada en el desarrollo e integración de software ágil indicó que las prácticas ágiles son igualmente aplicables en proyectos con SOA.

A continuación se mencionan algunas conclusiones obtenidas luego de la aplicación de Agile sobre SOA mediante una entrevista realizada a Geoff Henton y Tom Stiehm [103104].

[102] http://en.wikipedia.org/wiki/Shlaer-Mellor
[103] http://www.infoq.com/articles/agile-soa-implementation
[104] http://www.digitalfocus.com/research/whitepapers_soa.php

Las prácticas ágiles permitieron:

- A los responsables del proyecto cambiar el pensamiento evitándoles pensar en un diseño completo y un plan de proyecto predictivo.
- Cambiar la dirección a mitad del proyecto viendo el efecto total de sus decisiones en los objetivos del proyecto.
- Adaptarse rápidamente a los cambios producidos en las regulaciones federales.
- Sacar provecho de los equipos de trabajo dadas sus asignaciones de tiempos mediante la definición de iteraciones extremadamente cortas (de 1 día para generación de documentación y mayor obtención de feedback del cliente y 2 semanas para desarrollo del sistema).
- Mejor seguimiento del trabajo y ciclos de feedback mediante planificación de versiones, planificación de iteración, kickoff de iteración, relevamiento de requerimiento y al final de cada iteración retrospectiva de proyecto.
- Contratos más flexibles utilizando el principio “keep it simple” de las metodologías ágiles. El mismo se aplicó por ejemplo a dos casos. El primero a no finalizar los WSDL hasta no esté puesto en producción (siendo ésta una sola versión del contrato) a diferencia de aquellos que los dejan estáticos luego de la etapa de diseño y que luego difícilmente se adapte a los cambios en requerimientos. El segundo caso a no traducir una funcionalidad en servicio a no ser que haya al menos dos consumidores para ese servicio dado que esto evita la tendencia de crearlos en base a especulaciones de uso futuro.
- Reducción del tiempo de desarrollo. Menciona una comparación entre el desarrollo con metodologías tradicionales y con las ágiles reduciendo el tiempo entre un 25% y 35%.
- Poner en producción servicios mientras se van construyendo los demás. Si bien es fácil mejorar servicios que no están en producción, si se versionan los que sí lo están también es posible.

SOA les permitió:

- Entregar en tiempo funcionalidad que servía simultáneamente a demandas internas y externas.
- Exponer funcionalidades como servicios con el objetivo de ser consumidos por 12 de sus Bancos de Crédito. Estos permitieron a su vez, importar fácilmente datos acerca del manejo de fondos que luego emplearon en el uso en sus negocios.
- Al demostrar flexibilidad incrementar la confianza del negocio en IT para el resto del proyecto.

Agile y SOA lograron la alineación de IT con el negocio, significando esto que el negocio decida qué es necesario desarrollar y el desarrollo decidir cómo construirlo. Así pudieron entregar soluciones que el usuario pudo usar, el negocio tuvo su retorno de

inversión y la gente que trabajó en el proyecto tuvo la satisfacción de poner en producción una aplicación útil. Todo esto permitió incrementar la productividad.

Por último se mencionan las claves del éxito para el desarrollo de SOA como:

- Comprender los procesos del negocio que el cliente necesita implementar.
- Comprender SOA y los patrones.
- La adopción de una plataforma SOA puede crecer tanto como crezca el portfolio de servicios.
- Mantener un feedback frecuente de los clientes y el equipo de desarrollo para que se den soluciones no bien aparezcan cosas por resolver.
- Por otra parte, los seis pasos desarrollados para su proyecto con Agile y SOA:
- Introducción a SOA y capacitación.
- Vínculo entre SOA y los métodos ágiles.
- Definir objetivos del negocio a largo plazo.
- Definir una plataforma de software inicial.
- Definir e implementar una aplicación auto contenida o una aplicación sub sistema como primer release.

Por otra parte, Carl Ververs un estratega SOA que trabaja en ThoughtWorks, Inc, sugiere en base a su experiencia, cómo aplicar prácticas ágiles sobre SOA. A continuación se agregan los puntos clave sugeridos[105]:

Utilizar grupos de entre 4 a 8 desarrolladores con perfiles variados. Efectuar reuniones de media o una hora al comienzo de cada día donde se puedan discutir lecciones aprendidas, diseños y prácticas y planificar acciones futuras. Tener a la vista en una pizarra los artefactos que se van modificando diariamente para que todos conozcan lo que sucede y se pueda tener un mejor control de los cambios.

Seleccionar un flujo de proceso de mediana complejidad de los negocios centrales de la compañía. No se intentará resolver todo el problema, solo una solución que haga algo que parezca útil al cliente.

Graficar los caminos simples del flujo dejando fuera todas las excepciones, errores y puntos de decisión. Implementar las primeras piezas, uniendo de a dos a los desarrolladores teniendo como criterio de selección a aquellos cuyas experiencias y perfiles sean los más diferentes. Luego construir un consumidor MQ o JMS que envíe mensajes y algún componente que le responda un xml con datos.

[105] Agile: The SOA Hangover Cure. http://carlaugustsimon.blogspot.com/

- Servicios con flujo de proceso básico.
- Agregar servicios de validación. Incluir algunos otros servicios básicos. Mezclarlos de forma sincrónica, asincrónica y request-reply basada en mensajes. Escribir primero los tests.
- Expandir la validación. Introducir un camino de excepción que puede requerir ataduras entre uno o más pares de consumidores-proveedores. En ese caso se verá que es fácil cambiar el comportamiento de la aplicación dejando a la mayoría interactuando. Se pueden agregar métricas reuniendo algunas piezas, por ejemplo: tiempos de latencia en las respuestas, throughput, cantidad de mensajes, etc.
- Agregar excepción de validación. Una vez implementado algunos servicios, éstos se pueden mejorar o agregar nuevos. Siempre será necesario la colaboración del cliente para construir lo que realmente desea. Mientras vaya creciendo la cantidad de servicios publicados y en base a la incepción se va a necesitar una infraestructura con estándares. Se deberán identificar servicios y formatos usados por varios participantes SOA. Los servicios luego serán fáciles de mantener, escalables y confiables. Esto atraerá más usuarios de servicios.

Otros planteos[106] sugieren algunos puntos de discusión entre las prácticas SOA y las Ágiles. A continuación se detallan dos de ellos:

- Algunos autores plantean que SOA implica diseño up-front mientras que Agile no. Si bien pone foco en diseñar primero un buen contrato, el punto está en la granularidad. En SOA no es necesario construir el diseño de todos los bloques constructivos con gran detalle. Si bien los servicios necesitarán su diseño éste se puede desarrollar de forma incremental e iterativa.
- En algunas ocasiones se indica que SOA favorece la división de equipos de acuerdo funcionalidades mientras que Agile a los equipos cross funcionales. Si bien se deben definir los servicios en base a las necesidades del negocio, esto no implica la división de los equipos con este criterio. Cuando un servicio está en manos de un equipo, el equipo necesita tener una visión cross funcional para dar lugar a todos los temas relacionados para que ese servicio aporte valor. Por otro lado se plantea una posible división de los equipos (quizás ágiles) de acuerdo a los límites del servicio.

Gregor Hohpe fue entrevistado por Brenda Michelson[107] acerca de SOA y Agile. Hohpe comparó a Agile y SOA con manzanas y naranjas haciendo referencia a que a SOA se lo ve como un estilo de arquitectura que enfatiza cómo los sistemas interactúan mientras que Agile es una solución para el desarrollo que enfatiza cómo interactúan las personas. En

[106] www.infoq.com . SOA and Agile: Friends or Foes?. Amir Elssamadisy. 14/04/2007.
[107] http://www.enterpriseintegrationpatterns.com/ramblings.html

esta vista, las prácticas ágiles pueden ayudar a los desarrolladores a pasar de la teoría SOA a la práctica. A su vez, éstas intervienen en la evolución, iteración, pruebas, aprendizaje y colaboración del cliente ayudando a construir un puente entre la brecha que existe entre la teoría SOA y las demandas del mundo real.

Se considera adecuado el uso de Agile en proyectos con SOA debido a que permite adaptarse rápidamente a los cambios y a que fomenta la comunicación y colaboración requeridos a nivel de empresa, donde tanto proveedor del servicio como consumidor manejan diferentes tiempos sobre varios proyectos y equipos. La implementación de SOA va más allá del desafío tecnológico, es un desafío de personas y de proceso. La colaboración puede ser totalmente nueva en departamentos que formalmente son dueños de todos los datos y de la lógica del negocio necesaria para entregar una aplicación monolítica. Por esto mismo, Agile provee un framework sólido acerca de cómo se debería interactuar para hacer las entregas lo antes posible.

Si se tiene una arquitectura SOA con Agile entonces se tendrá una organización que rápidamente identifique los cambios necesarios en el software y una unidad de desarrollo que rápidamente implemente esos cambios[108].

En respuesta[109] al artículo publicado por Brenda Michelson se indica que la compatibilidad entre Agile y SOA depende de varios factores:

- De acuerdo a si se está teniendo en cuenta la construcción de SOA utilizando técnicas ágiles, ó si se está construyendo con otra metodología con el la perspectiva de permitir agilidad una vez que haya suficiente cantidad de servicios disponibles para hacer la diferencia.
- Si se han puesto en prácticas los métodos ágiles, de qué forma, si hay casos de éxito, etc.
- Según es estado de los métodos ágiles en la organización, si las técnicas son nuevas para el negocio, o si hay experiencias sustanciales con éstos métodos.
- Los métodos ágiles son acoplados mientras que SOA requiere bajo acoplamiento.

Por otra parte explica que cada organización tiene su curva de aprendizaje con una nueva metodología de desarrollo al igual que Agile. En algunas compañías la adopción de Agile podría no ser exitosa si poseen una cultura con malos hábitos, como el estar constantemente buscando soluciones mágicas y descartando aquello que no le da resultados rápidamente. Por esto mismo, indica que las técnicas ágiles no salvarán a una organización con capacidades de ejecución pobres.

[108] http://certifiedarchitect.blogspot.com/2007/01/agile-soa-take-2.html
[109] http://enterprisearchitect.typepad.com/ea/2006/07/agilty_and_soa_.html

Una SOA bien diseñada permitirá agilidad a futura con respecto al desarrollo de software. Recomienda que sería mejor iniciar SOA con una metodología con una administración de proyecto más tradicional si es que Agile es nueva para el negocio. Esto le permitirá ganar experiencia y avanzar en la construcción de SOA y luego podrá ser posible aplicar Agile en el desarrollo del software.

Por último comenta un acerca de caso donde un colega intentó implementar desde el inicio SOA con Agile (una capa de abstracción de datos que abarque muchas bases de datos desde las aplicaciones mediante la invocación de servicios web para poner y sacar datos de las bases) pero fracasó. La falta de experiencia en Agile, el tener que estar continuamente educando a los desarrolladores, el refactor constante de los servicios web para cumplir con los cambios en los requerimientos, degradaron los servicios web debido a que el acoplamiento se tornó cada vez mayor a ésta aplicación específica.

5.5 Comparaciones y análisis exploratorio

En esta sección se presentará un cuadro comparativo acerca de las metodologías de desarrollo aplicables en SOA estudiadas en la sección anterior. Luego, se analizará el resultado de una encuesta, realizada a profesionales de sistemas de nuestro país, con el objetivo de conocer acerca los proyectos SOA y del uso de las mismas.

5.5.1 Características básicas

Aquí se presenta un análisis comparativo entre las metodologías de desarrollo de software en Arquitecturas Orientadas a Servicios. Las características que se muestran en este cuadro, reflejan de forma sintética lo estudiado en las secciones anteriores.

Las características comparadas son las que se listan a continuación.

- Etapas del ciclo de vida que cubre, si participa de todas o solo de un conjunto de ellas.
- Delivery Strategy. Qué estrategia de delivery SOA implementa: Top-Down, Bottom-Up, ó Meet-In-The Middle.
- Artefactos entregables. Aquí se indicará si la metodología especifica los artefactos a entregar en cada fase de desarrollo.
- Adopción de otras técnicas. Si la metodología se basa en otras metodologías y/o hace uso de técnicas ó prácticas como por ejemplo UML, BPM, UDDI, Análisis y Diseño Orientado a Objetos, etc.
- Casos de éxito reportados. Si se han publicado casos de estudio que indiquen el resultado de su utilización en proyectos SOA.

- Soporte a equipos distribuidos. Si indica técnicas o métodos como soporte para el trabajo con equipos distribuidos físicamente.
- Prácticas ágiles. Si especifica o recomienda el uso de prácticas ágiles para el desarrollo de software. Se decidió incluir este punto debido a la importancia que se vio de éstas prácticas en la sección de Metodologías Ágiles.
- Control de Calidad. Si interviene ó reporta técnicas para el control de calidad.
- Propietario. Si se trata de una metodología propuesta por algún proveedor de software particular y por lo tanto de las especificaciones no son abiertas, ó si se trata de una metodología abierta donde se encuentra a disposición toda la documentación necesaria para analizar sus capacidades, sus aportes, sus ventajas y desventajas.

El cuadro comparativo se encuentra en la sección de Anexos, específicamente en la sección 8.1.

Este cuadro también se utilizará para elaborar las conclusiones del presente trabajo.

5.5.2 Encuesta

Para esto y con el objetivo de analizar a modo exploratorio, cuál es la tendencia que se presenta hoy día en cuanto a ellas, se efectuará el análisis de una encuesta realizada a personas involucradas en proyectos con SOA. Este estudio no pretende ser concluyente sobre qué tipo de metodologías son las mejores, así como tampoco asumir que las respuestas obtenidas de estas entrevistas representan la totalidad de los perfiles posibles de participantes de un proyecto SOA. Lo que se pretende es obtener de primera mano las sensaciones del usuario en la participación de proyectos SOA y con el uso, o no, de una metodología de desarrollo como guía de proyecto.

La encuesta comienza preguntando si se conoce el concepto de SOA y si ha participado en proyectos con ésta arquitectura. En caso afirmativo, se pregunta acerca de experiencias vividas en ellos. Si se empleó un registro UDDI, si experimentaron ventajas y desventajas luego de la implementación SOA, si el ROI fue positivo, si se contaba con un SOA Competence Center, etc. Se intenta conocer acerca de la experiencia del encuestado, con metodologías de desarrollo de software. El objetivo es conocer si se utiliza ó utilizó alguna metodología, si se ha capacitado a los usuarios antes de su aplicación, si implementan prácticas ágiles y si las mismas aportaron buenas prácticas o estándares en SOA.

También se pregunta al encuestado si tiene experiencia en la aplicación de herramientas de integración, propietarias u Open Source.

El propósito no es conocer qué metodología en particular utilizan, sino conocer si utilizan alguna metodología, y en el caso de hacerlo, qué características a grandes rasgos presenta.

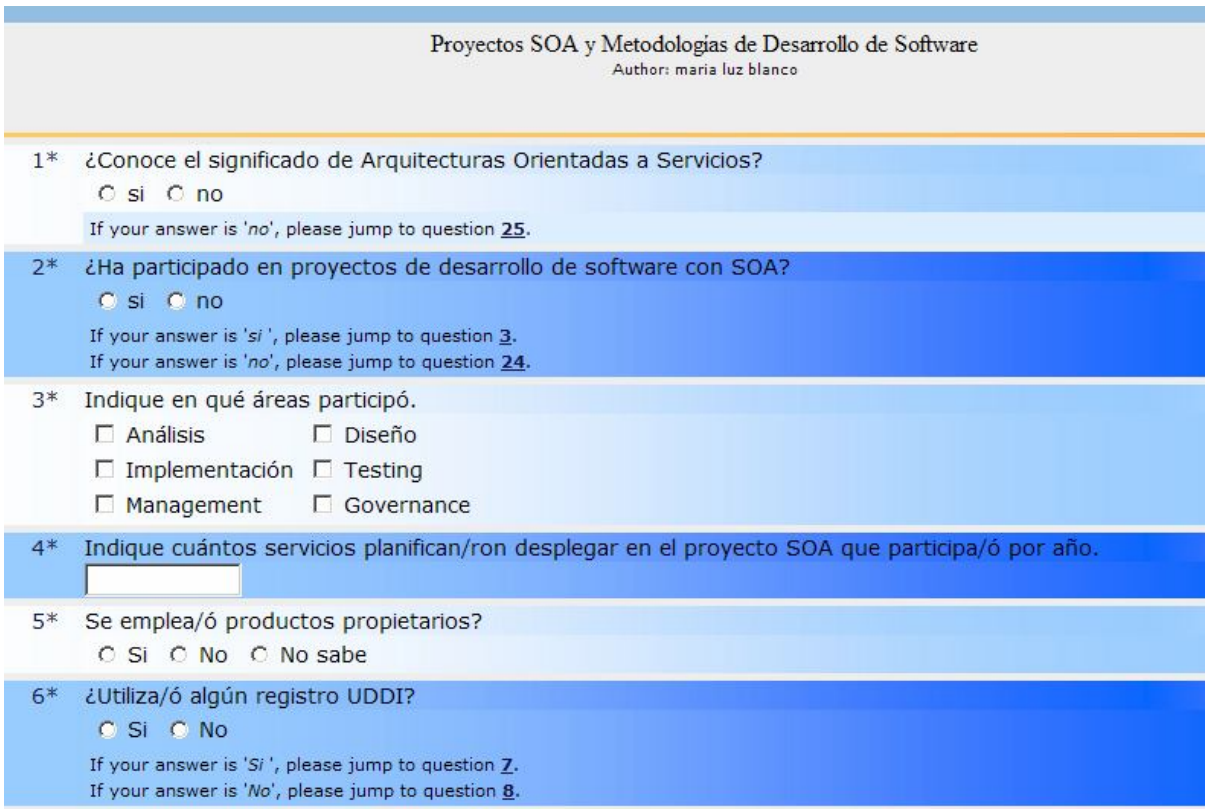
Por último, se deja con casillero para que el encuestado complete con los comentarios que considere necesarios, y en el caso de querer una copia del presente trabajo, se le pide que deje su correo electrónico.

La encuesta se completará desde Internet, utilizando un servicio de la Web 2.0 en la página www.my3q.com. Esta herramienta permite crear entrevistas de forma gratuita, facilitando la recolección y análisis de los datos que se carguen en ellas.

Se ha solicitado que completen la encuesta, a más de 40 profesionales de sistemas de información, que se encuentran actualmente trabajando en su profesión. La distribución fue mediante el envío por mail de la dirección donde se encuentra alojada la encuesta. La mayoría de estas personas participan de proyectos SOA. Además, fue distribuida a un grupo de desarrolladores java alojado en yahoo Argentina, cuyo nombre es “DesarrolloJava”, (http://ar.groups.yahoo.com/group/DesarrolloJava/) y el cual está compuesto por más de 2500 miembros. El tamaño de la muestra obtenido es de 31 personas.

La encuesta consiste en preguntas directas con respuestas de selección simple o múltiple. Solamente en tres puntos se permitió al encuestado desarrollar libremente la respuesta ingresando texto en un casillero. Estos puntos fueron agregados al final de la encuesta con el objetivo de conocer un poco más acerca de la experiencia del encuestado.

La encuesta consta de las siguientes preguntas:



Proyectos SOA y Metodologías de Desarrollo de Software

Author: maria luz blanco

1* ¿Conoce el significado de Arquitecturas Orientadas a Servicios?

○ si ○ no

If your answer is '*no*', please jump to question **25**.

2* ¿Ha participado en proyectos de desarrollo de software con SOA?

○ si ○ no

If your answer is '*si* ', please jump to question **3**.
If your answer is '*no*', please jump to question **24**.

3* Indique en qué áreas participó.

☐ Análisis ☐ Diseño
☐ Implementación ☐ Testing
☐ Management ☐ Governance

4* Indique cuántos servicios planifican/ron desplegar en el proyecto SOA que participa/ó por año.

5* Se emplea/ó productos propietarios?

○ Si ○ No ○ No sabe

6* ¿Utiliza/ó algún registro UDDI?

○ Si ○ No

If your answer is '*Si* ', please jump to question **7**.
If your answer is '*No*', please jump to question **8**.

7 ¿Considera que el uso de UDDI fue de utilidad?
○ Si ○ No ○ No sabe

8 Indique si la aplicación de SOA aportó alguna de éstas ventajas.
☐ Reuso ☐ Flexibilidad
☐ Interoperabilidad ☐ Seguridad
☐ Orquestación ☐ Escalabilidad
☐ Reducción de costos ☐ Agilidad
☐ Reducción de riesgos

9 ¿Han experimentado alguna de éstas limitaciones en la implementación de un proyecto con SOA?
☐ Overhead
☐ Escaso soporte en la especificación de políticas y contratos
☐ Falta de una herramienta específica de versionado
☐ Falta de una metodología que guíe el desarrollo de software
☐ Un mal diseño provoca gran impacto en sistemas con los que se vincula
☐ Costos elevados
☐ Complejidad
☐ Falta de personal idóneo.

10 ¿Cuentan con algún comité de governance SOA?
○ Si ○ No ○ No sabe

11 ¿Tuvieron/tienen en cuenta antipatrones SOA y cómo evitarlos?
○ Si ○ No

12 ¿Conoce si el Retorno de Inversión (ROI) fue positivo con la implementación de SOA?
○ Si ○ No

13* ¿Ha participado alguna vez en algún proyecto de desarrollo de software que haya utilizado alguna metodologías de desarrollo de software?
○ Si ○ No
If your answer is '*Si* ', please jump to question **14**.
If your answer is '*No*', please jump to question **15**.

14* ¿Tuvo capacitación previa acerca de la/s misma/s?
○ Si ○ No

15* ¿Conoce alguna metodología de desarrollo de software aplicable a proyectos con SOA?
○ Si ○ No
If your answer is '*Si* ', please jump to question **16**.
If your answer is '*No*', please jump to question **24**.

16* Si Ud. ha participado de un proyecto con SOA, ¿Utiliza/ó alguna/s metodología/s de desarrollo de software?
○ Si ○ No ○ No sabe
If your answer is '*Si* ', please jump to question **17**.
If your answer is '*No* ', please jump to question **24**.

17* ¿Qué fases del ciclo de vida de un proyecto de software toma en cuenta esa/s metodología/s?
☐ Planeamiento ☐ Análisis ☐ Diseño
☐ Construcción ☐ Testing ☐ Deployment
☐ Governance

18* La/s metodología/s empleada/s, ¿recomienda/n estándares, buenas prácticas, estrategias para lograr el éxito con SOA?
○ Si ○ No ○ No sabe

19* ¿Qué estrategia de delivery implementa?
☐ Top-down ☐ Meet-in-the-middle
☐ Bottom-up

20* ¿La/s metodología/s que utiliza/n es/son?
○ Tradicionales ○ Híbridas
○ Ágiles

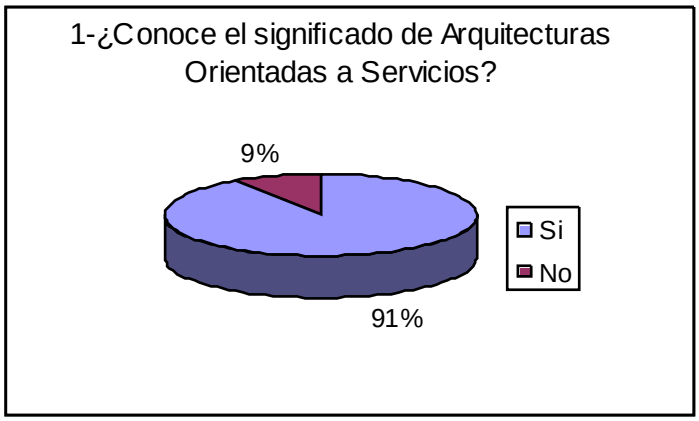
21* ¿Encuentra a su disposición material de soporte ó consulta sobre ella/s?
○ Si ○ No ○ No sabe

5.5.2.1 Análisis de resultados

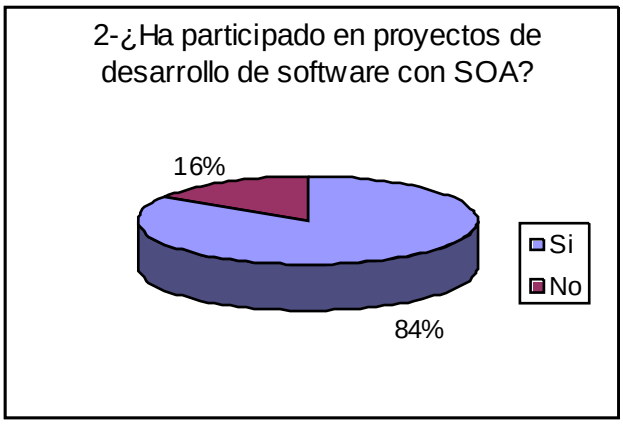
En esta sección se hará un análisis exploratorio sobre el resultado obtenido para cada respuesta presente en la encuesta.

El objetivo de la primera pregunta es saber si el encuestado conoce acerca del concepto de Arquitecturas Orientadas a Servicio. Básicamente esta pregunta se ha incluido en el cuestionario porque si el encuestado no conoce acerca de las mismas, entonces no podrá contestar el resto de las preguntas. Por esto mismo, se indica que si no se conoce, deberá pasar directamente a la pregunta 25, salteando las anteriores.

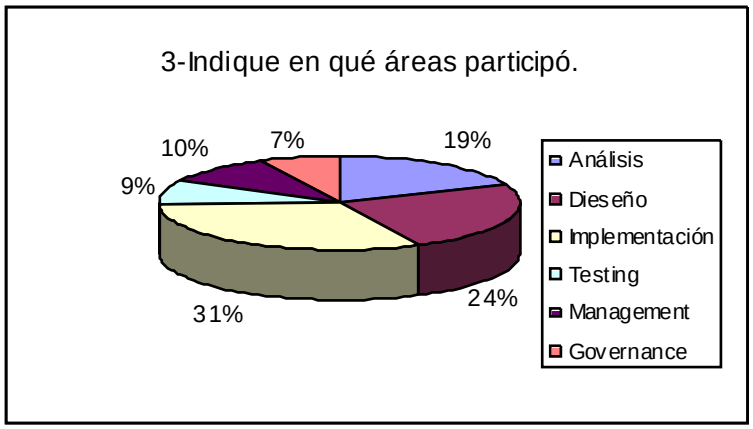
Esta pregunta indicó que el 91% sabía acerca de SOA, y solo un 9% respondió lo contrario. Esto indica, que si bien la encuesta se distribuyó a personas relacionadas con el mundo de sistemas, todavía existe un porcentaje importante que no conoce acerca de éste tipo de arquitectura. Ya hemos indicado anteriormente, que este tipo de arquitecturas es fundamental para alcanzar la alineación de IT con el negocio dentro de una organización.



La segunda pregunta apuntaba a conocer si el encuestado participó o participa en proyectos donde existe SOA. Esta pregunta es importante debido a que, lo que se esperaba conseguir en esta encuesta, es el aporte de personas con experiencia en proyectos con SOA para los cuales se encuentren desempeñando tareas en diferentes áreas. El 84% contestó afirmativamente, esto indica que las respuestas siguientes aportarán mayor valor cuando se pregunte acerca de las experiencias vividas en el mundo SOA. Por otra parte, indica que el porcentaje de personas involucradas en el mundo SOA es cada vez mayor.

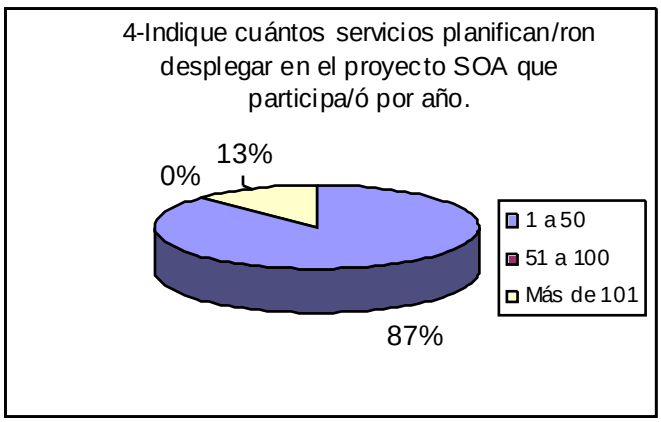


La tercera pregunta se propuso conocer acerca del perfil del encuestado. Específicamente en qué etapas del ciclo de vida del desarrollo de software participó. La mayor parte respondió participar en la implementación o construcción de sistemas. El siguiente porcentaje correspondió a la fase de diseño y en tercer lugar al análisis. Testing, Management, Governance, fueron las áreas con porcentajes más parejas y pero con menos porcentaje.



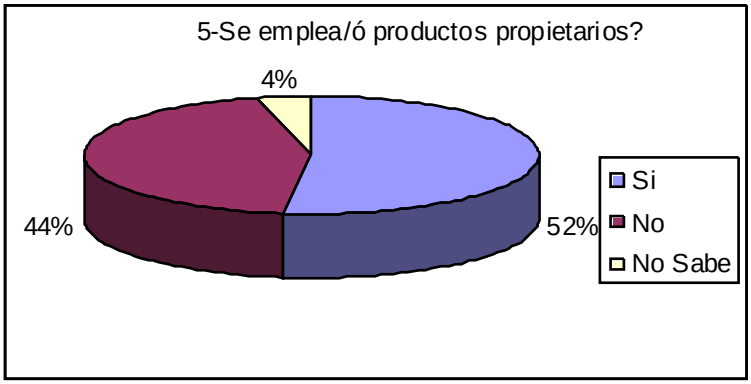
La cuarta pregunta apuntó a detectar qué cantidad de servicios se planificó o planificaba desplegar por año. Este es un indicador del tamaño de proyecto en el cuál participa ó participó el encuestado. Se pondrá especial atención en este detalle, debido a que a mayor cantidad de servicios, se requerirá mayor orquestación, mejor diseño, prácticas ágiles, análisis de antipatrones y cómo evitarlos, entre otras cosas.

Los resultados se agruparon en tres categorías: de 1 a 50, de 51 a 100 y de 101 servicios en adelante, encontrándose el mayor porcentaje en el primer rango, 0% en el segundo y un 13% en el tercero. Esto podría darnos indicaría que el tamaño de proyecto en que se basó el encuestado, es pequeño o mediano, si bien se registraron respuestas donde se indicó la planificación de aproximadamente 200 servicios al año. Este último caso podría estar relacionado a proyectos que corresponden a empresas grandes donde se ha optado por adoptar SOA como arquitectura, incorporando varios servicios existentes, y otros nuevos, para obtener el mayor beneficio en cuanto a la alineación de su negocio y IT.

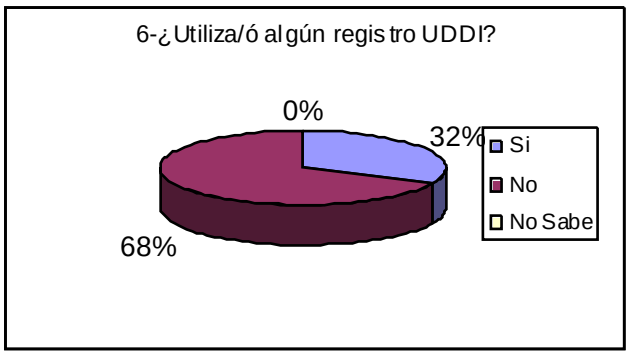


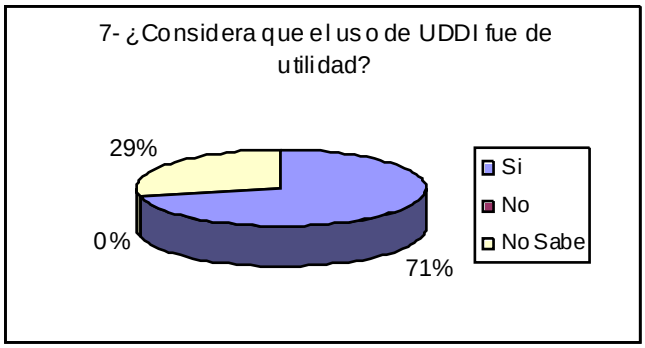
Con la quinta pregunta se intenta saber si en los proyectos donde el encuestado participó, se emplearon productos propietarios. El objetivo puntual de ésta pregunta está relacionado con lo visto durante las secciones anteriores, cuando se señaló que gran parte de la motivación que se muestra hoy día, está relacionada con vendedores de productos

aplicables en SOA. Sin embargo, si bien la encuesta indicó con un 52% que se utilizaban productos de éstas características, un 44% señaló lo contrario.

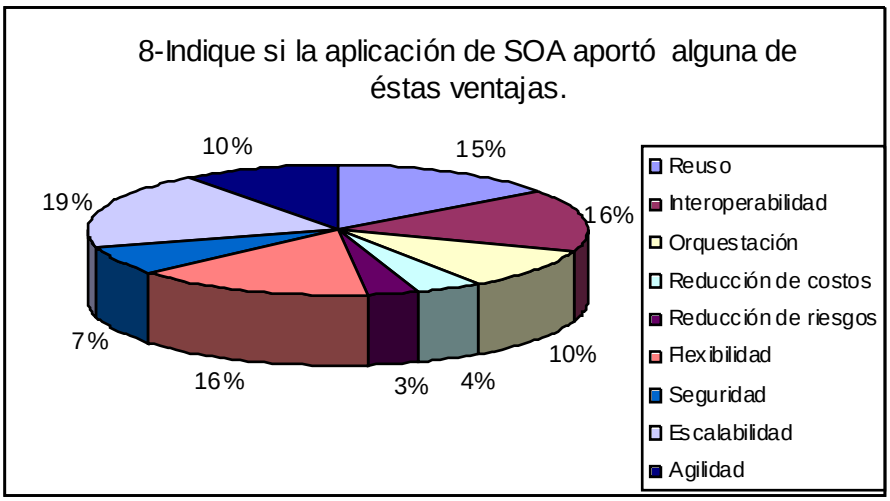


La siguiente pregunta apunta a conocer en qué porcentaje se opta por emplear un registro UDDI. Esta encuesta indicó con un 68% que no se los han empleado. Teniendo en cuenta la pregunta 4, esto podría deberse a que, los proyectos en los que el encuestado participó, son pequeños o medianos. En muchas organizaciones, frente a estos casos, la utilización de un registro UDDI podría ser considerada como esencial, hasta que se tenga desplegado una mayor cantidad servicios. En esos casos ya se ha visto que el uso de un registro de servicios es crítico. Por otra parte, quizás en éstos proyectos se utilizan otras herramientas que desempeñen de forma más reducida y sencilla la funcionalidad de UDDI. En algunos casos, se opta por utilizar una simple planilla con los servicios disponibles, manteniendo la descripción de los mismos, su dirección, su contrato, etc. Un dato importante relevado aquí mismo es que ningún encuestado respondió no conocer acerca de la implementación de UDDI en proyectos SOA. Esto indica positivamente, el conocimiento de UDDI y su posible aplicación en un proyecto, con las ventajas y limitaciones que esto representa. En relación con esta pregunta, la siguiente indicó con un 71% que la implementación de UDDI fue de utilidad. A continuación se incluyen los dos diagramas que lo representan.





Las preguntas 8 y 9 están destinadas a conocer acerca de las ventajas y limitaciones que aportó SOA en los proyectos de los que participó el encuestado. Teniendo en cuenta las ventajas de adopción e implementación de SOA, la que más se destacó fue la Escalabilidad con un 19%, a continuación y con un 16% la Flexibilidad e Interoperabilidad y con muy poca diferencia, con un 15% el Reuso. Orquestación y Agilidad con un 10%.

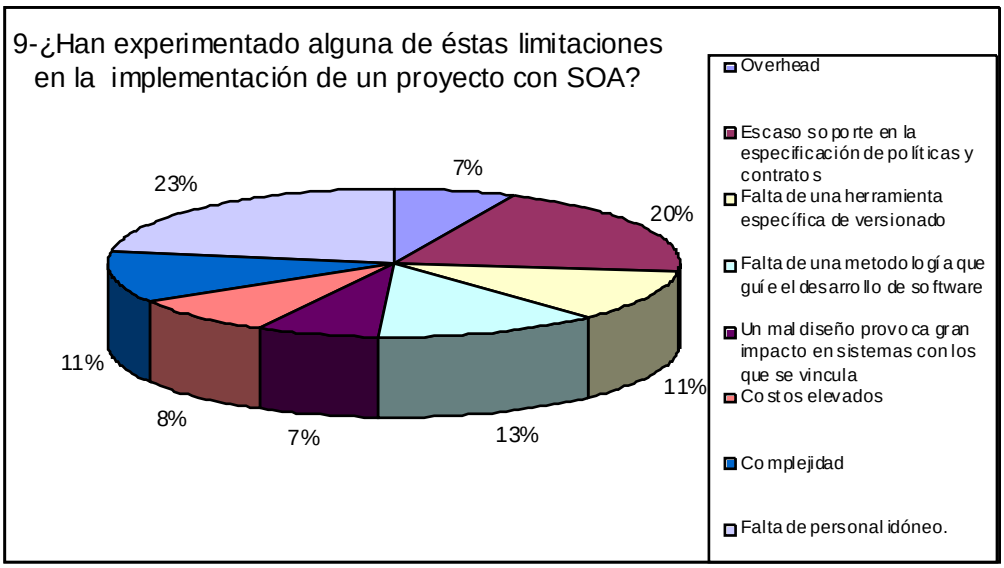


Para el caso de las desventajas que aportó SOA, la falta de personal idóneo fue la más indicada con un 23%. Esto indica, una vez más, que SOA se encuentra en los primeros pasos, que se encuentra en crecimiento, y que los actores involucrados deben capacitarse para lograr un mejor desempeño y aportar mayor valor en un proyecto con éstas características. Esto mismo se reflejó, en esta encuesta, en el punto 25 donde muchas personas señalaron la falta de conocimiento y la necesidad de personal calificado apto para trabajar en proyectos de éstas características.

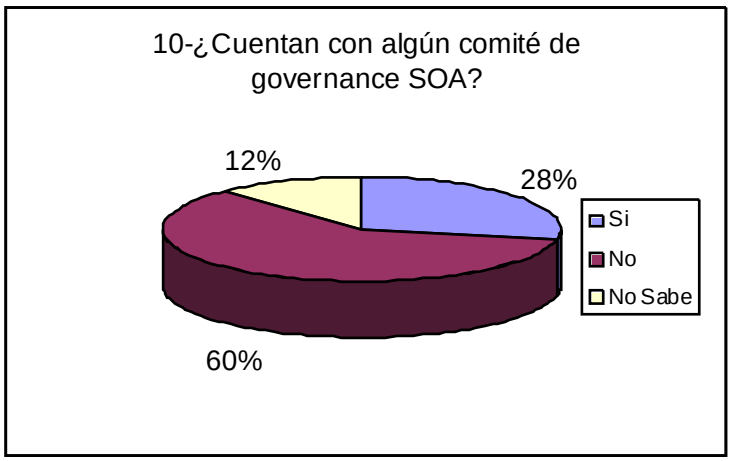
En segundo lugar y con un 20% se indicó el escaso soporte en la especificación de políticas y contratos. Tal como se vio antes, una de las principales características relacionadas con los servicios en SOA es la especificación de los mismos, la definición de una interfaz apropiada de forma que ésta permita el mayor reuso, escalabilidad, representatividad del mismo, etc. El alto porcentaje en el soporte relacionado con la especificación de políticas y contratos es muy importante y señala un aspecto crítico en el que SOA debe madurar.

El resto de las desventajas poseen porcentajes muy similares.

Las preguntas relacionadas con las metodologías de aplicables en proyectos SOA ayudarán a comprender mejor éstos resultados.

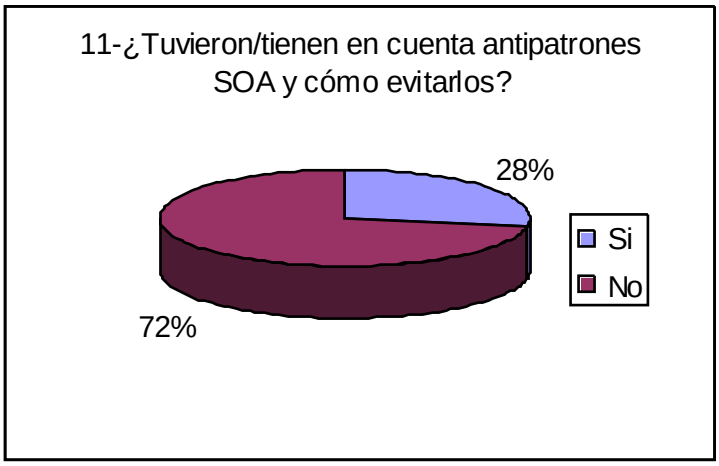


En la pregunta 10 se cuestiona si en la organización donde el encuestado forma ó formó parte, poseen un comité SOA. Sólo el 28% indicó afirmativamente. Esto puede estar relacionado con el tamaño del proyecto, ó la fase en la que éste se encuentre. En proyectos pequeños que se encuentran en las primeras fases de SOA, se ha notado la ausencia de un Comité SOA, a pesar de haber estudiado la importancia del Governance como actividad clave en la gestión de SOA.

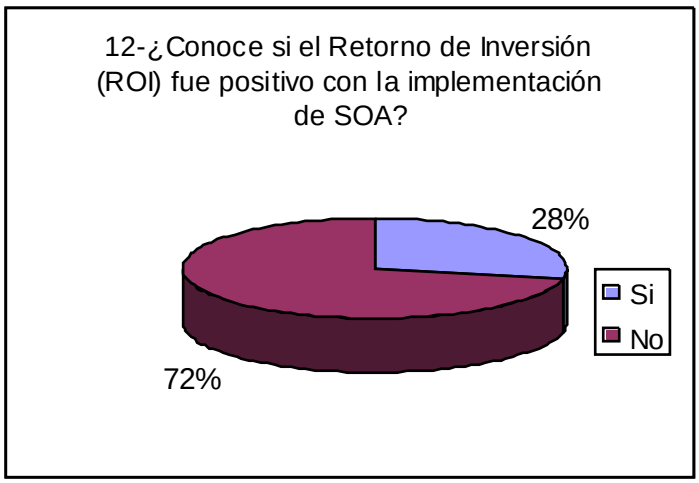


La pregunta 11 hace referencia a si se tienen en cuesta antipatrones en los proyectos y la forma de evitarlos. Ya se ha visto que el diseño en SOA es una de las fases más importantes, y el tener en cuenta patrones de diseño es una buena práctica a aplicar.

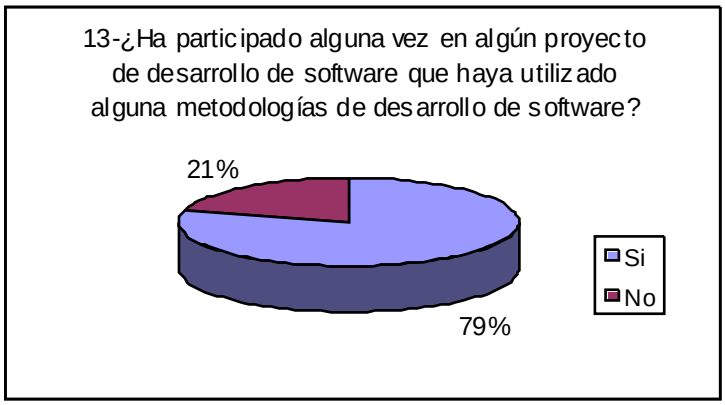
También se han indicado metodologías que indican la forma de evitarlos. No obstante esto, se ha observado que solo el 28% ha notado que en los proyectos esto se tiene en cuenta.



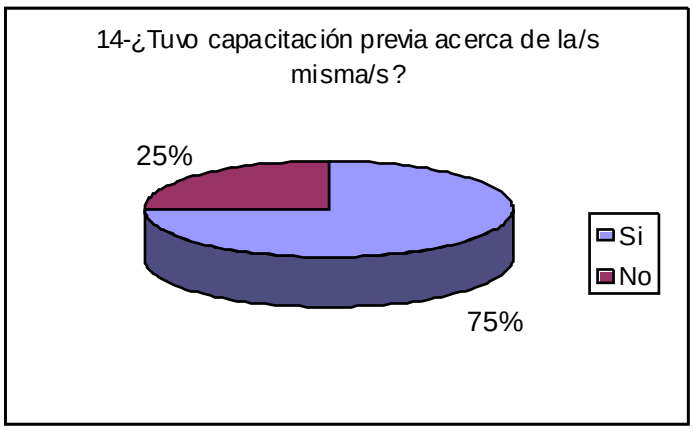
El resultado arrojado en esta encuesta indicó que en relación a la pregunta 12, el 72% de los encuestados respondió desconocer si el Retorno de Inversión fue positivo. Esto puede deberse, entre otras cosas, a que el proyecto se encuentra en una fase inicial, ó a que simplemente el encuestado participa en áreas donde no le es posible conocer este dato.



El 79% de los encuestados indicó haber participado en proyectos donde se aplicó metodologías de desarrollo de software. A continuación, se intentará conocer la experiencia del encuestado en relación a este tema.

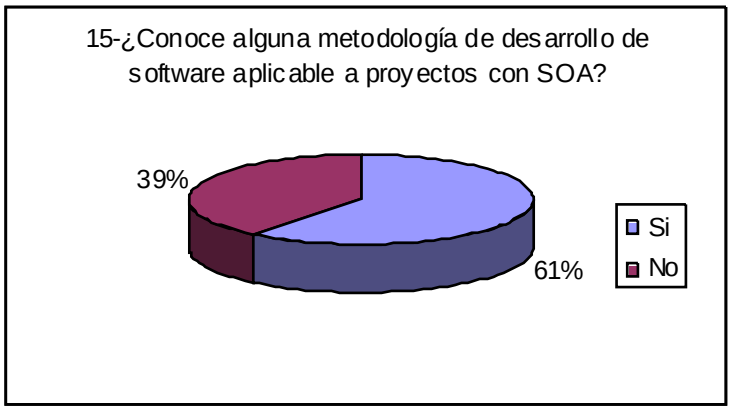


Afortunadamente, los encuestados que participaron del desarrollo de software aplicando alguna metodología, indicaron haber tenido capacitación previa a la utilización de las mismas.

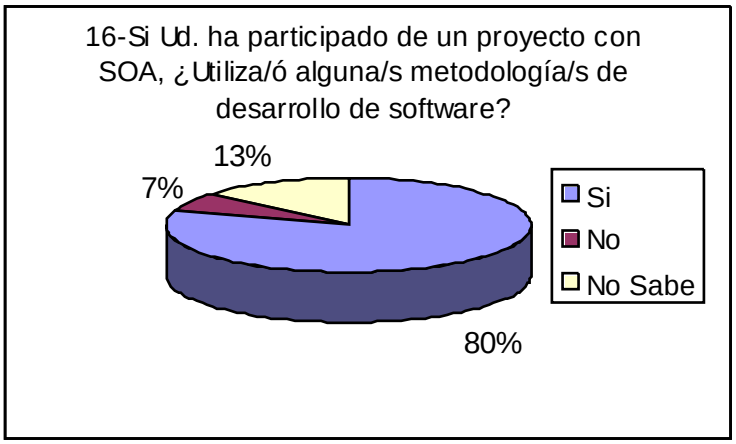


A continuación se analizan las preguntas relacionadas con la aplicación de metodologías de desarrollo en proyectos con SOA. Se aclara que el objetivo de las mismas fue conocer si realmente se aplica una metodología (y no cuál), de qué tipo es, y si da soporte a SOA.

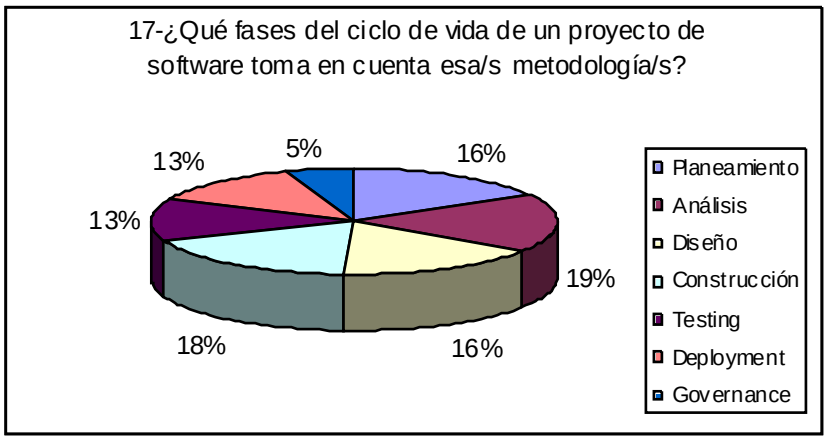
El resultado de la siguiente pregunta indicó con un 61% que los encuestados conocen metodología de desarrollo aplicables a SOA. Este alto porcentaje podría estar relacionado al punto 13 donde se vio que la mayoría tiene experiencia con el trabajo guiado por una metodología. Y tal como se ha estudiado en secciones anteriores, se sabe es posible aplicar las metodologías de desarrollo tradicionales, híbridas ó ágiles en proyectos SOA, si bien se recomienda hacer ajustes sobre ellas. En conclusión, quizás el encuestado no solo esté haciendo referencia a las metodologías específicas para proyectos SOA sino que también a otras conocidas en proyectos con otras arquitecturas subyacentes.



A su vez, con la pregunta 16 se pudo observar que el 80% de los encuestados participaron de proyectos con metodologías de desarrollo en proyectos SOA. Es llamativo saber que el 13% indicó no saber si en el proyecto que participa se utiliza alguna metodología. En un 7% se detectó que no se aplican metodologías de desarrollo. En este trabajo se ha indicado la importancia del uso de alguna metodología.

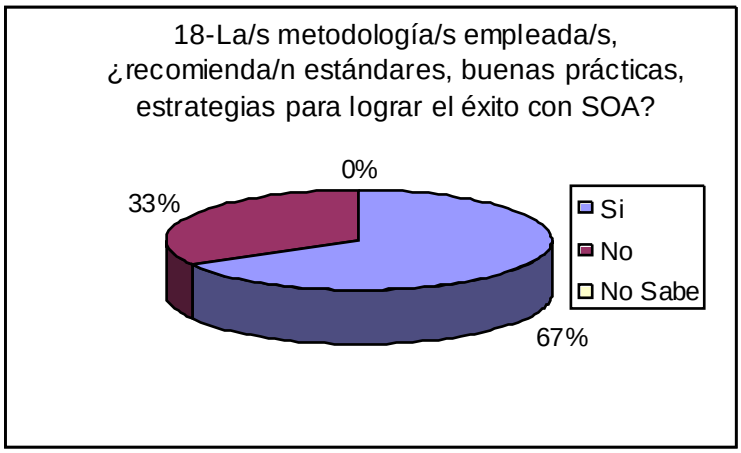


De las metodologías que se han utilizado en proyectos SOA se indicó que las mismas tuvieron en cuenta las fases de análisis, construcción, diseño y planeamiento.

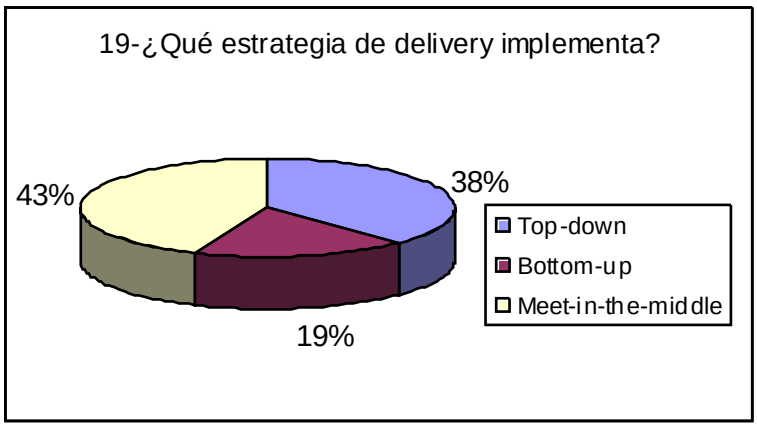


El resultado de la siguiente pregunta, afortunadamente indicó que las metodologías empleadas sugirieron estándares, buenas prácticas y estrategias para lograr el éxito con SOA. Si bien el porcentaje es amplio, un 67, el 30% indicó lo contrario. Esto demuestra que

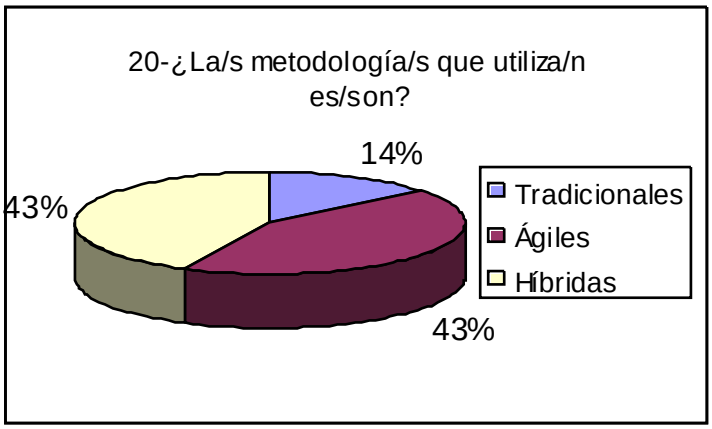
es necesario un replanteamiento de las metodologías que se emplean en proyectos con estas características. Especialmente si se utilizan las metodologías empleadas en otros proyectos con una arquitectura diferente. En ese caso, quizás sea necesario hacer un estudio más a fondo de los aportes a SOA ó de nuevas metodologías que sean más acordes a SOA. A pesar de esto, también hay que reconocer que, debido a que las metodologías aún no han publicado variedad de casos de éxito, ó experiencias que indiquen que su uso, las empresas no tienen aún el soporte suficiente, necesario para optar por un cambio.



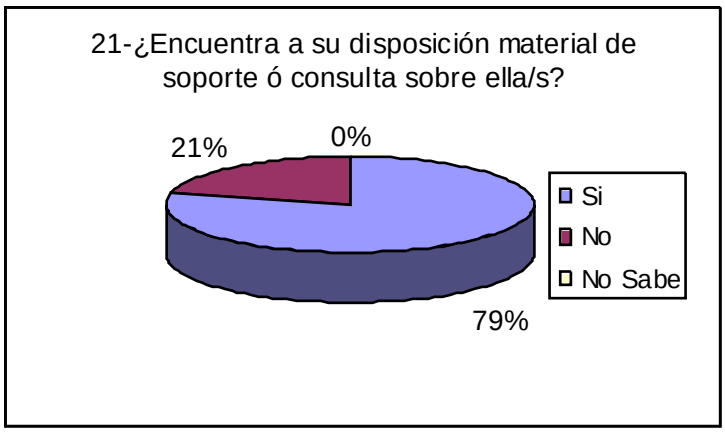
La pregunta 19 se incluyó en el cuestionario para conocer la estrategia de delivery según las metodologías utilizadas. La estrategia más utilizada observada fue Meet-In-The-Middle, luego Top-Down y por último Bottom-Up. Este alto porcentaje da noción acerca del tipo de empresas en las cuales se está implementando SOA. En ellas se intenta lograr un balance entre los servicios que se generan a partir de las funcionalidades del negocio y aquellos ya existentes, por ejemplo, la integración de sus sistemas legacy en funcionamiento.



La pregunta 20 reflejó, con un 43% que las metodologías más utilizadas son híbridas o ágiles en igual medida. En segundo lugar, y con casi la mitad del porcentaje, se encuentran las tradicionales.



Tal como se ha visto anteriormente, es muy importante contar con información disponible acerca de la metodología que se utiliza en un proyecto. Si bien se ha visto, durante la realización de éste trabajo que algunas metodologías poseen poca información disponible acerca de ellas, el resultado de esta pregunta indicó con un 79% que de alguna forma, han tenido a su disposición el material necesario.



A continuación se incluirán comentarios que indican ventajas de las metodologías utilizadas, publicados por algunas personas encuestadas en respuesta a la pregunta 24. Se hará referencia a los mismos en las conclusiones de éste trabajo.

"Procesos ordenado y orientados al servicio. Definición de interfases y contratos".

"Principalmente flexibilidad".

"El reuso también es visible, aunque muchas veces es mas criterio de quienes lo implementan".

"Reusabilidad, Interoperabilidad, estar atado a plantear y pensar los contratos de antemano, bien definidos, escalabilidad".

"Personalmente me parece que se logran mejores resultados utilizando dos metodologías: Una tipo RUP para hacer la planificación y definición de requerimientos e implementación (opcional) de servicios divisionales y corporativos. Estos servicios son los que se utilizan desde otras divisiones y generalmente son utilizados por grupos de desarrollos distintos, por lo que resulta útil que todos los grupos esten al tanto de los contratos de estos servicios, cambios en los mismos y planificación de implementaciones. Luego, para desarrollar los servicios de mas bajo nivel me parece mucho mas

efectivo utilizar una metodología ágil tipo Scrum, pues esta dentro de un mismo grupo de desarrollo y por lo general por mejor planificado que este se requieren frecuentes cambios y ajustes menores, que encajan mucho mejor en esta metodología".

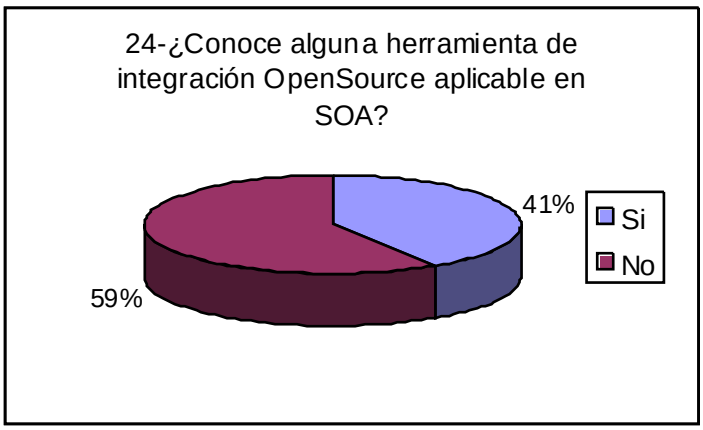
" Dificultad en métodos consensuados para realizar testing y generar puntos de control válidos".

"Relacionado con el punto 22, me parece que SOA requiere una metodología estricta y abarcativa para ciertos servicios (los de mas alto nivel) y una metodología ágil para los servicios de más bajo nivel, por lo que una metodología sola no encaja con todos los tipos de servicio".

También se incluyen algunos comentarios donde se indicaron desventajas luego de la aplicación de metodologías de desarrollo de software sobre proyectos SOA.

"Falta de Soporte".
" Ausencia de un standard homogéneo y concertado en la comunidad".
"Inestabilidad en las herramientas".

Por último y como para evaluar si el encuestado conoce herramientas de integración tipo open source aplicables a SOA, se obtuvo como resultado, un 59% de respuestas afirmativas. Esto indica que si bien el mayor porcentaje de ellos conoce éste tipo de herramientas, existe un alto porcentaje que indicó lo contrario.



El punto 25 de ésta encuesta, presentó un casillero donde el encuestado podría dejar un comentario ó consignar su e-mail para enviarle el presente trabajo. Vale mencionar que gran parte de ellos lo hizo indicando que les gustaría recibir una copia motivados por su interés en conocer más acerca de SOA y las metodologías de desarrollo que se pueden aplicar en proyectos donde se tenga este tipo de arquitectura.

Este detalle no es menos importante, debido a que, como se ha visto anteriormente, si bien existe un alto porcentaje de personas que siendo profesionales de sistemas, no conocen acerca de SOA, existe la intención de aprender acerca de la misma.

5.5.3 Conclusiones.

Tal como se ha mencionado al final del estudio de las metodologías de desarrollo de software, no es posible determinar una mejor metodología en el desarrollo de sistemas con arquitecturas orientadas a servicios. Como se ha visto anteriormente, existe actualmente una variedad importante de metodologías aplicables al desarrollo de éste tipo de arquitecturas. Sin embargo, debido a que se trata de una arquitectura con recientes aplicaciones, la mayoría de éstas metodologías pertenecen a empresas desarrolladoras de software que han construido metodologías propietarias y que intentan venderlas junto con sus productos. Si bien algunas de ellas aportan recomendaciones básicas para la implementación de arquitecturas SOA, la mayoría aún carece de suficiente información que las avale y de casos de éxito válidos como para optar por su adopción.

La encuesta mostró con sus resultados, que las metodologías de preferencia por las organizaciones hoy día, en nuestro país, son las ágiles y las híbridas. Este punto es importante debido a que SOA se desarrolla en un marco donde los cambios se producen frecuentemente y por lo tanto, se requiere de técnicas y prácticas que permitan adquirir la flexibilidad necesaria para poder abrazarse al cambio.

Quizás, tal como lo sugirió Kunal Mittal [MITTAL-SOA] la opción recomendada sería utilizar un híbrido, tomando de cada metodología las mejores prácticas y teniendo en cuenta no caer en los antipatrones SOA. Mittal sugiere utilizar RUP en la primera fase “Rollout inicial SOA”, donde se construye la arquitectura, ya que posee funcionalidades ya construidas para soportar el desarrollo de SOA y los servicios web. Para la segunda etapa “Mantenimiento SOA” , sugiere utilizar XP ya que considera que es una metodología liviana diseñada para entregar software de acuerdo a las necesidades de los usuarios cuando éstos lo requieren y responde correctamente ante los cambios en los sus requerimientos, aún en etapas avanzadas del ciclo de vida. Además, indica que es un método rápido para capturar requerimientos asignando prioridades y desarrollando historias de usuarios. Sin embargo cree que no cubre la validación de requerimientos con lo cual si la calidad en éstos no es lograda los resultados podrían no ser satisfactorios.

La realización del análisis exploratorio de la encuesta permitió observar, una vez más que el mundo SOA se encuentra en pleno desarrollo. Muchos proyectos ya han planificando el despliegue de gran cantidad de servicios bajo SOA, sin embargo otros todavía están recién en el comienzo, adoptando SOA de a poco, de forma iterativa e incremental.

De acuerdo a la experiencia de los encuestados, la mayoría participó de proyectos con SOA donde se utilizó una metodología top-down o meet-in-the-middle. Por su parte, un gran porcentaje si bien indicó poseer la información necesaria, reportó la falta de soporte disponible ó de casos de éxito que ayuden verificar el beneficio que las mismas aportan a cada proyecto.

Por otra parte, tal como se explicaba anteriormente teniendo en cuenta la opinión de algunos encuestados y de acuerdo con la postura de Mittal, existe una gran cantidad de personas que recomiendan el uso de una metodología tradicional para las primeras fases del ciclo de vida de desarrollo de un proyecto con SOA y desde la fase de implementación en adelante, utilizar una ágil como ser Scrum ó XP.

6. LINEAMIENTOS RESCATADOS Y VISION A FUTURO.

SOA es un tipo de arquitectura, que si bien ha surgido hace unos años, todavía se encuentra en crecimiento. La adopción a SOA consiste en un proceso evolutivo del cual forman parte grandes cambios en la gestión tecnológica, cambios en la organización, culturales y metodológicos. Partiendo de esto, y teniendo en cuenta la opinión de la consultora Gartner[110] en base a SOA, el 70% de las organizaciones de TI, en sus primeros intentos no alcanzarán el éxito. No obstante esto, en este estudio, se han incluido las ventajas, limitaciones y también desafíos de SOA. Las metodologías de desarrollo de software aplicables a SOA, deben ser un factor esencial para lograr el éxito con SOA y superar los desafíos que hoy día se plantean.

Con respecto a las metodologías, se espera que cada una de ellas también evolucione; tenga en cuenta el ciclo de vida completo de desarrollo de un proyecto con SOA, ponga a disposición de los responsables de proyectos información de consulta acerca de los beneficios que puede aportar como guía de desarrollo de software, e indique experiencias de implementación que los ratifiquen.

Con respecto a SOA se espera que los costos de implementación, sobre todo en las primeras fases, se reduzca. Que en cada proyecto se implementen comités SOA, que mantengan una política clara a nivel empresarial, que evalúen las arquitecturas existentes y las metodologías de forma tal que se ajusten apropiadamente a un proyecto SOA, que fijen un mapa de ruta con el objetivo de aportar mayor valor al negocio, etc. Que evolucionen las herramientas de integración y testing tipo open source, la especificación de estándares y políticas para la definición y regulación del uso de servicios, las buenas prácticas para evitar antipatrones, entre otras cosas.

Además es importante que se formen profesionales aportando valor en todas las fases del ciclo de vida del proyecto.

Con respecto a los trabajos, la encuesta realizada a modo exploratorio en este trabajo, podría volver a realizarse nuevamente dentro de un tiempo en el país, para obtener algún parámetro de referencia de la evolución de la experiencia SOA en las organizaciones. Por otra parte, podría realizarse también en otros países con el objetivo de realizar comparaciones acerca del desarrollo de esta arquitectura en ellos y en nuestro país.

[110] http://www.gartner.com/

7. GLOSARIO

Diseño Up-Front

La siguiente definición ha sido tomada de Wikipedia[111] en el enlace indicado en el pie de página[112].

Un diseño Up-Front ó Big Design Up Front, es una metodología de desarrollo de software donde el diseño del programa debe ser completado y perfeccionado antes del comienzo de la implementación del mismo.

Quienes defienden esta metodología sugieren que el tiempo que se pasa en la planificación es una inversión valiosa y citan casos en que ésta metodología demostró insumir menor tiempo y esfuerzo en resolver una falla en las primeras etapas del ciclo de vida del desarrollo de software que en cuando el mismo error se encuentra luego y es necesario corregirlo. Con esto sugieren que es más sencillo corregir un requerimiento en fase de requerimientos que en fase de implementación, como corregir un requerimiento en fase de implementación que requiere pelear con la implementación y el trabajo de diseño ya completado.

Por otra parte, quieren están en contra de ésta metodología, generalmente provienen de los métodos ágiles e indican que es muy poco adaptable a los cambios en requerimientos, que es desactualizada e inválida, y que asume que los diseñadores pueden predecir y tienen la capacidad de prever áreas con problemas sin un prototipo ni inversión en una implementación.

Método

Pressman [PRESSMAN-94], definió método en el contexto de la ingeniería de software como métodos que proveen líneas guía técnicas para el desarrollo del software. Método constan de un conjunto de tareas que incluyen:

- planeamiento y estimación del proyecto.
- Análisis de requerimientos de sistema y de software.
- Diseño de datos estructurado.
- Arquitectura de programa y algoritmos procedimentales.
- Codificación.
- Pruebas.
- Mantenimiento.

[111] http://wikipedia.org/
[112] http://en.wikipedia.org/wiki/Big_Design_Up_Front

Por método se entiende entonces una colección de técnicas y un conjunto de reglas para describir cómo y en qué orden deben ser ejecutadas para alcanzar determinados objetivos [AGILE-T&P].

Metodología.

Generalmente Metodología y Método son tomados como sinónimos, pero en este trabajo será tomado como la ciencia o estudios del método.

Según Wikipedia, la rama de la metodología, dentro de la ingeniería de software, se encarga de elaborar estrategias de desarrollo de software que promuevan prácticas adaptativas en vez de predictivas; centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega, de comunicación intensiva y que requieren implicación directa del cliente.

Ortogonalidad

En el diseño de software, la ortogonalidad es la habilidad de cambiar una parte conceptual del sistema de software mientras se reducen los efectos colaterales en otra parte. Un diseño ortogonal separa diferentes cuestiones del sistema como ser: acceso a datos, seguridad y lógica de negocio de forma que permita ser aislado de áreas de código sin tener que entrelazarse.

Los conceptos básicos para la escritura de código ortogonal son el débil acoplamiento y la alta cohesión.[113].

Proceso Pesado

Proceso Pesado o Heavy process en inglés, son términos que se utilizan para sugerir alguna de las cualidades de procesos de software que se mencionan a continuación [DIET-PROCC]:

- Rigidez y control.
- Muchas actividades y artefactos creados en una atmósfera burocrática.
- Pérdida de documentos.
- Elaboración de planes detallados a largo plazo.
- Sobrecarga en muchos procesos en la cima del trabajo esencial.
- Orientación a procesos en lugar de orientación a las personas, tratamiento de personas como artefactos que se conectan en un método mecánico.
- Predictivo en lugar de Adaptativo.

[113] http://codebetter.com/blogs/jeremy.miller/archive/2007/01/08/Orthogonal-Code.aspx

Proceso Predictivo

Proceso Predictivo o en inglés Predictive Process, se denomina al proceso que intenta planear y predecir las actividades y la ubicación de recursos humanos en detalle a largo plazo como en la mayoría de los proyectos. Tiende a ser implícitamente en cascada en su valor, definiendo enfáticamente los requerimientos en primera instancia y el diseño detallado en segunda, antes de la implementación [RUP-FAIL].

Proceso Ágil

Proceso ágil también llamado en inglés Agile Process ó Light Process en Inglés, posee las siguientes características [RUP-FAIL]:

- Despojado de sobrecarga burocrática innecesaria, eliminando la creación de documentación sin valor.
- Está enfocado en las realidades de la naturaleza humana en el contexto del trabajo, haciendo que el desarrollo del software sea agradable.
- Es adaptativo.

Safety-critical software[114]

También llamado Life-Critical software, son sistemas en los cuales sus fallas o mal funcionamiento pueden ocasionar:

- muerte o serios daños a personas,
- pérdida, o daños severos en equipamiento,
- daños ambientales.

Sistema Legacy

Según Wikipedia[115], se utilizan éstos términos para hacer referencia a un sistema informático habitualmente anticuado (ordenador o aplicación) que continúa siendo utilizado por el usuario (típicamente una organización) y no quiere o puede ser reemplazado o actualizado.

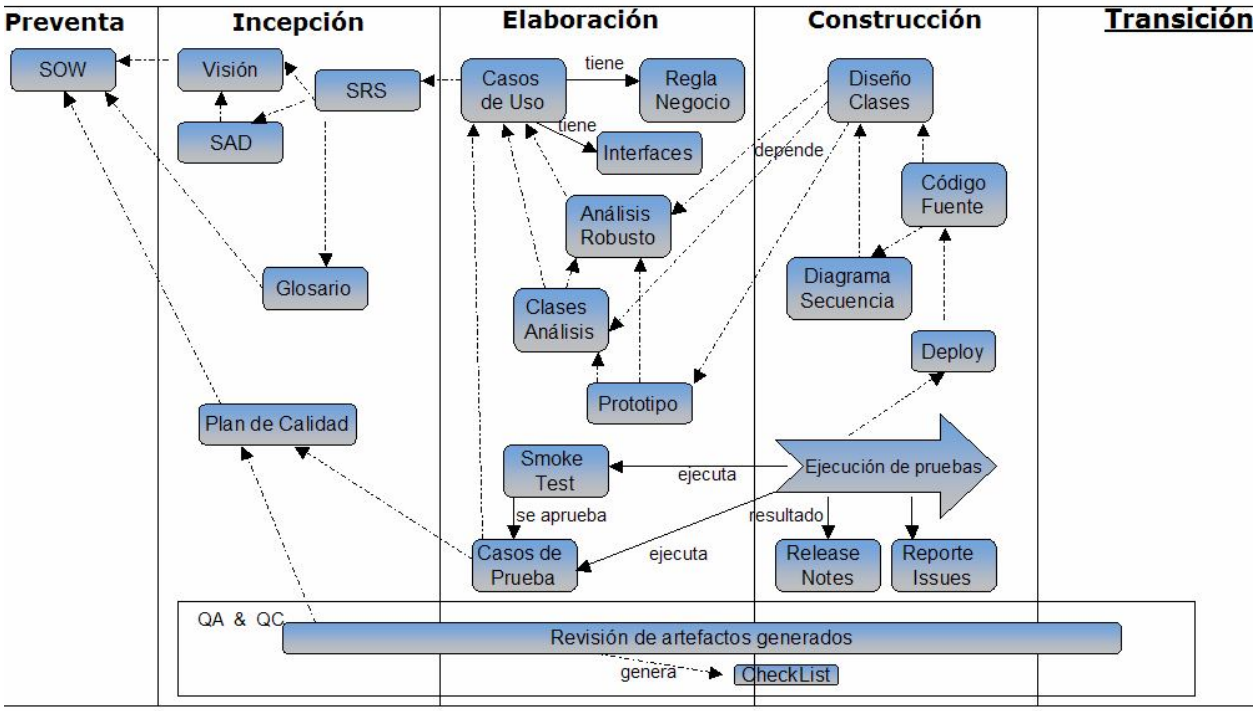
[114] http://en.wikipedia.org/wiki/Life-critical_system
[115] http://www.wikipedia.org/

8. ANEXOS

8.1 Comparación de metodologías de desarrollo en SOA

	Etapas del ciclo que vida que cubre.	Delivery Strategy	Disponibilidad Información	Artefactos a entregar	Adopción de otras técnicas	Casos de éxito reportados	Soporta el trabajo con equipos distrib.	Utiliza prácticas ágiles	QA	Propietario
Ágiles	Todas de acuerdo al tipo de metodología ágil	Todas de acuerdo al tipo de metodología ágil	Alta	Código, casos de prueba	UML, OO, en algunos casos	Si	Si	Si	Dirigido por casos de prueba	No
RUP	Incepción, Elaboración, Construcción, Transición	No especifica	Alta	De acuerdo a la fase.	UML	Si	Si	Agile RUP si	QA y QC	Si
IBM - SOAD	Análisis y Diseño	Meet-in-the-middle para procesos	Alta	Casos de Uso. Contratos	OOAD, ESB, SOAP, WSDL, UDDI, BPM	Si	No especifica	No especifica	Factores de calidad	Si
IBM - SOMA	Análisis y Diseño (Identificación de servicios candidatos y flujos, especificación de servicios, componentes y flujos y realización)	Todas de acuerdo al objeto que se está identificando.	Alta	Artefactos de Análisis y Diseño de RUP más un modelo de servicios	OOAD, UML, UDDI	No	No especifica	No especifica	Identifica antipatrones y la forma de evitarlos.	Si
SUN - RQ	Concepción, Incepción, Elaboración, Construcción, Transición.	Top-down	Baja	Templates entregables.	UML, RUP, XP	No	No especifica	Si	No especifica	Si
CBDI-SAE	Planificación, análisis y diseño, construcción, pruebas, deployment, actividades de governance	Meet-in-the-middle	Media	No especifica	UML	No	No especifica	Si	No especifica	No
SOAF	Elicitación del servicio, identificación, definición del servicio, realización, roadmap y planificación	Meet-in-the-middle	Baja	No se encontró información al respecto	No se encontró información al respecto	No	No se encontró información al respecto	No se encontró información al respecto	No se encontró información al respecto	No
SOUP	Incepción, Definición, Diseño, Construcción, Deploy y Soporte.	Meet-in-the-middle	Media	De acuerdo a la fase.	RUP para primera fase, luego XP. UML	No	No especifica	Si	Testing. No especifica detalles	No
Steve Jones	planificación, administración y delivery. Identificación de servicios. Creación de SOA. Desarrdlo de la arq. completa Adm. de cambios	Top-down	Alta	De acuerdo a la fase.	No especifica	No	No especifica	Si	No especifica	No
XP	Exploración, Planificación, Iteraciones, Producción, Mantenimiento y Muerte del proyecto	-	Alta	Código, SRC, Historias de usuario	No especifica	Si	Técnicas como video-conferencias	Si	Si. Tests Unitarios	No
BEA	Soporte en el Ciclo de vida de un servicio: Identificación, Descomposición , Deploy y Governance. Modelo de Dominio BEA.	-	Alta	Contratos	No especifica	Si	No especifica	Feedback continuo.	No especifica	Si

8.2 Mapa de Artefactos RUP.



9. BIBLIOGRAFÍA

[AGIL-DACS] Agile Software Development. A DACS State-of-the-Art Report. David Cohen, Mikael Lindvall and Patricia Costa. Fraunhofer Center for Experimental Software Engineering Maryland and The University of Maryland. Rome, New York. 31/01/2003.

[AGILE -ECO] Agile Software Development Ecosystems. Jim Highsmith. Addison Wesley. 25/05/2002.

[AGILE-ACOCK] *Agile Software Development.* Alistair Cockburn. New York. Año 2001.

[AGILE-ISSI] Metodologías Ágiles en el Desarrollo de Software. Ingeniería del Software y Sistemas de Información. Taller realizado en el marco de las VIII Jornadas de Ingeniería del Software y Bases de Datos, JISBD 12/11/2003.

[AGILE-LIM] Limitations of Agile Software Processes. Dan Turk, Robert France, Bernhard Rumpe. Colorado State University y Munich University of Technology respectivamente.

[AGILE-METH] Agile software development methods. Review and Analysis. Pekka Abrahamsson, Outi Salo, Jussi Rokainen & Juhani Warsta. VTT Publications. 2002.

[AGILEM-XP] Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). Patricio Letelier y María Carmen Penadés. Universidad Politécnica de Valencia.

[AGILE-PRACT] On the productivity of Agile Software Practices: An Industrial Case Study. F. Maurer y S. Martel. Año 2002.

[AGILE-QES] What Is Agile Software Development?. Jim Highsmith. Agile Software Development. Octubre de 2002.

[AGILE-REQ] Requirements Engineering in Agile methods. Hossein Horrian, Shafquat Mahmud, Srinivasan Karthikeyan. Dept. of Computer Science, University of Calgary. Dept. of Electrical & Computer Engineering, University of Calgary, Canada.

[AGILE-SOA] The Agile/SOA Potential. Contributed by Liz Barnett. 09/07/2006.

[AGILE-SURV] 2nd. Annual Survey. "The State of Agile Development". VersionOne y APLN. Junio - Julio de 2007.

[AGILE-T&P] Agile Software Development in theory and practice. Jonna Kalermo y Jenni Rissanen. Universidad de Jyvaskyla. Añ0 2002.

[AMR-REP] Service-Oriented Architectures: Survey Findings on Deployment and Plans for the Future. AMR Research Report. Eric Austvold y Karen Carter. Año 2005.

[B2B-TRENCH]B2B Integration Trends: Message Formats. Alternatives Grow, But EDI Standards Remain The Leading Option For B2B Messaging. Ken Vollmer, Mike Gilpin y Jacqueline Stone. 06/07/2007.

[BASILI-TURNER] V. Basili and J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," IEEE Trans. Software Eng., Diciembre de 1975.

[BEA-DMOD] Domain Model For SOA. Realizing the Business Benefit of Service-Oriented Architecture. BEA White Paper. Julio de 2005.

[CASC-VS-ITER] A Comparative Study of Iterative Prototyping vs. Waterfall Process Applied To Small and Medium Sized Software Projects. Eduardo Málaga Chocano. National University of Engineering, Lima, Peru. Abril 2004.

[CASES-AGILESD] A case-study based assessment of Agile software development. William Herman Morkel Theunissen. University of Pretoria etd. Año 2003.

[CBDI-REP] CBDIReport. CBDI-SAETM. CBDI Service Architecture & EngineeringTM. Everware-CBDI. Octubre de 2006.

[Conferencias] XP Agile Universe: www.agileuniverse.com. Conference on eXtreme Programming and Agile Processes in Software.

[DEF-SOA] SOA: Qué es realmente. Óscar Roncero. Progress Software. Marzo de 2007.

[DEM-SOA] Demystifying SOA. Jared Rodríguez Chief Technology Officer. Skyway Software Inc. August 2006. www.skywaysoftware.com.

[DIET-PROCC] Put Your Process on a Diet in Software Development. Fowler, Martin. CMP Media. Diciembre 2000.

[DSDM-PRINC] Dynamic System Development Meted. Benjamin J. y J. Voigt. Department of Information Technology. University of Zurich. 02/01/2004

[DSDM-ZURICH] Dynamic System Development Meted. Benjamin J. J. Voigt. Department of Information Technology University of Zurich. Suizam, 20/01/2004.

[EAI-LINTH] Enterprise Application Integration. David S. Linthicum. Addison Wesley Professional. 12/11/1999.

[EAI-LINTH2] Next Generation Application Integration: From Simple Information to Web. David S. Linthicum. Addison Wesley. 15/08/2003.

[ERL-DSTRA] SOA Delivery Strategies. Service Oriented Architecture: Concepts, Technology, and Design. Erl, Thomas. 08/02/2005.

[ERL-SOAC] Service-Oriented Architecture: Concepts, Technology, and Design. Thomas Erl. Prentice Hall. 04/10/2005.

[ESTB-RUP] Estructura básica del proceso unificado de desarrollo de software. Robin Alberto Castro Gil. Universidad Icesi. 14/04/2004.

[FC-BOHTU] Balancing Agility and Discipline: A Guide for the Perplexed. B. Boehm y R. Turner. Boston. Addison Wesley. Año 2003.

[FC-JONES] Software Assessments, Benchmarks, and Best Practices. C. Jones. Boston. Addison Wesley. Año 2000.

[FDD-PALMER] S Palmer. Feature Driven Development and Extreme Programming. Coad Letter. www.togethercomminity.com/coad-letter/Coad-Letter-0070.html

[IBM-AGILE] Service-oriented agility: Methods for successful Service-Oriented Architecture (SOA) development, Part 1: Basics of SOA and agile methods. IBM developerWorks. http://www-128.ibm.com/developerworks/webservices/library/ws-agile1/

[IBM-PATTERNS] Patterns: SOA Foundation Service Creation Scenario. John Ganci, Amit Acharya, Jonathan Adams, Paula Diaz de Eusebio, Gurdeep Rahi, Diane Strachan, Kanako Utsumi y Noritoshi Washio. Septiembre 2006.

[IBM-SOMA] Moving ahead with SOA. Bussiness services especification with SOMA: customer experiences feedback. Olivier Dennery. IBM. Año 2005.

[IBM-TERM] SOA terminology overview, Part 3: Analysis and design. Bertrand Portier. IBM. 16/052007.

[ICONIX-PROC] Agile Development with ICONIX Process—People, Process, and Pragmatism. Doug Rosenberg, Matt Stephens y Mark Collins-Cope. Apress. Año 2005.

[IID-BHISTORY]Iterative and Incremental Development: A Brief History. Craig Larman (Valtech), Victor R. Basili (University of Maryland). Published by the IEEE Computer Society. June 2003.

[IMAZ] Derribando mitos. Manuel Imaz. 9/11/2006. http://www.bloggers.com.ar/system/noticia_detalle.php?id_prod=676

[JACKSON-JDM] The Jackson Development Method. Wiley Encyclopedia of Software Engeneering. Año 1992.

[JACKSON-JSD] Jackson System Development. Overview. http://www.dsisoft.com/jackson_system_development.html

[JACKSON-SDM] A System Development Method. Michael A. Jackson. Michael Jackson Systems Limited. Londres.

[JONES-METH] A methodology for service architectures. Steve Jones y Mike Morris. Londres. Capgemini UK plc. 26/10/2005.

[LEAN-POPP] Lean Software Development: An Agile Toolkit for Software Development Managers. Mary Poppendieck y Tom Poppendieck. Año 2006.

[LEAN-SOA] Service-oriented agility: Methods for successful Service-Oriented Architecture (SOA) development, Part 2: How to mix oil and water. Gottrried Luef, Christoph Steindl y Pal Krogdahl. IBM. 09/08/2005.

[METH-MARKS] Development Methodologies Compared. Why different projects require different development methodologies. Dan Marks. Diciembre 2002. www.ncycles.com

[METH-SRVAR] A methodology for service architectures. Steves Jones. Mike Morris. 26th October 2005. Capgemini UK Capgemini 2005

[MICROSF-LC] Microsoft Architect Journal. Service-Oriented Architecture: Considerations for Agile Systems. Lawrence Wilkes y Richard Veryard. CBDI Forum. Abril de 2004.

[MITTAL-SOA] Build your SOA, Part 1: Maturity and methodology. Kunal Mittal. 20/05/2005. http://www.soaconsultant.com.

[MS-RAD] Rapid Development. McConnell Steve. Microsoft.1996: pp. 141-143.

[NEWM-FOW] The New Methodology. Martin Fowler. 13/12/2005.

[OASIS-RM] OASIS Reference Model for Service Oriented Architecture. Committee Specification 1,2. August 2006.

[OIT-ASD] Office of Information Technology, Agency Software Division, Software Development Methodology Reference. State of new Hampshire. 18/02/2005.

[PRESSMAN-94] Software Engineering: a practitioner´s approach (adaptado por Darrel Ince). 1994. Ed. McGraw-Hill.

[QUOVADX] The business case for SOA. White paper. Quovadx, Inc. www.roguewave.com

[RCG SOA] RCG Information Technology. Service Oriented Architecture. www.rcgit.com.

[RUP-AGILE] The Agile Unified Process. Sinan Si Alhir, PMP, IT Project. 13/03/2005.

[RUP-BESTP] Rational Unified Process Best Practices for Software Development Teams. A Rational Software Corporation White Paper. 1998 Rational Software Corporation.

[RUP-CASES] Using Rational Unified Process in an SME – A Case Study. Geir Kjetil Hanssen, Hans Westerheim y Finn Olav Bjørnson. Norwegian University of Science and Technology. Agosto 2005.

[RUP-ELECT] Electronic Process Guides in Connection With the Use of RUP at ConsultIT. Lars Roar Næsset & Amar Bhargava. Departement of Computer and Information Science, NTNU. Otoño de 2003.

[RUP-FAIL] How to Fail with the Rational Unified Process: Seven Steps to Pain and Suffering. Craig Larman, Philippe Kruchten, Kurt Bittner. 14/07/2002. Valtech Technologies & Rational Software.

[RUP-INTRO] The Rational Unified Process: An Introduction, Third Edition. Philippe Kruchten. 19/12/2003.

[RUP-MANAG] A Manager's Introduction to The Rational Unified Process (RUP). Scott W. Ambler. 04/12/2005.

[RUP-PGUIDE] Then Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Per Kroll, Philippe Kruchten. Addison Wesley. 11/04/2003.

[RUP-PKR] What Is the Rational Unified Process?. Philippe Kruchten. Rational Fellow. Rational Software Canada. Enero 2001.

[RUP-PMREP3] Project Management report 3. The Evaluation of RUP/UP software development process. Maria Chitul, Vassili Garkusha, Przemyslaw Rudzki & Serghei. Blekinge Institute of Technology.

[RUP-VAL1] Rational Unified Process (RUP). Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia. https://pid.dsic.upv.es

[SCRUM-PRIM] The Scrum Primer. An Introduction to Agile Project Management with Scrum. Pete Deemer y Gabrielle Benefield. Año 2007.

[SCRUM-SCHW] SCRUM Development Process. Advanced Development Method. Ken Schwaber. 15/08/2006.

[SCRUM-STEAM] The Scrum Software Development Process for Small Teams. Linda Rising y Norman S. Janoff, AG Communication Systems. Julio-Agosto 2000.

[SERI-REV] SERIKAT, Consultoría e Informática. Andreka Puentes.

[SOA-ADOBE] Service Oriented Architecture. Whitepaper. Duane Nickull. Adobe Systems, Inc. USA. Año 2005.

[SOA-BATUTA] Proyecto Batuta - Generador de Aplicaciones Orquestadoras. Pablo Alvez, Patricia Foti, Marco Scalone. Facultad de Ingeniería - Universidad de la República. Uruguay. Junio 2006.

[SOA-BPPH] Enterprise SOA, Service Oriented Architecture Best Practices. Dirk Krafzig, Karl Banke y Dirk Slama. Prentice Hall. Año 2005.

[SOA-ELEM] Elements of Service–Oriented Analysis an Design, an interdisciplinary modelling approach for SOA Projects. Olaf Zimmermann, Pal Krogdahl y Clive Gee. 02/06/2004.

[SOAGILITY] Service Oriented agility: Methods for successful Service-Oriented Architecture (SOA) development, Part 1: Basics of SOA and agile methods. Pal Krogdahl, Gottrried Luef y Christoph Steindl. 26/07/2005.

[SOA-IMPLEM] Implementing a Service-Oriented Architecture – More Than Just Web Services. Andrew Price y Mike Sander. Mayo de 2005.

[SOA-LFQA] SOA Lifecycle Quality: The “Tipping Point” of SOA Governance. LISA whitepaper series. Jim Mackay. Copyright 2007 iTKO, Inc. 16/06/2006.

[SOA-LIFEC] Comparing Software Development Life Cycles. Jim Hurst. 23/03/2007. http://www.giac.org/resources/whitepaper/application/257.php

[SOAP-NET] Building Web Services and .NET Applications. Wall & Lader. Capítulo 7. Año 2001.

[SOA-REDBOOK] SOA transition Scenarios for the IBM z/OS Platform. Alex Louwe Kooijmans, Raymond Chiang, Irin Litman, Mats Pettersson, Bill Seubert, Jens Erik Wendelboe. International Technical Support Organization. SOA Transition Scenarios for the IBM z/OS Platform. Marzo de 2007.

[SOA-RUPXP] Suitability of Extreme Programming and RUP Software Development Methodologies for SOA Applications. Guillermo A. Callahan. Helsinki University of Technology. Finlandia.

[SOA-SOC] Service-Oriented Architecture (SOA) vs. Component Based Architecture. Helmut Petritsch.

[SOA-SURV] A Survey of Service Oriented Development Methodologies. Ervin Ramollari, Dimitris Dranidis, and Anthony J. H. Simons. South East European Research Centre y Department of Computer Science. University of Sheffield.

[SOA-UY] Desarrollo de aplicaciones con enfoque SOA (Service Oriented Architecture). Andrea Delgado, Laura González, Federico Piedrabuena. Universidad de la República, Facultad de Ingeniería, Instituto de Computación, Montevideo, Uruguay.

[SOAWP-HP] Service Oriented Architecture White Paper. Manoj Mansukhani. Hewlet Packard. www.hp.com/go/SOA. 28/06/2005.

[SOA-WPRAC] SOA Worst Practices, Volume I. How to prevent your SOA from being DOA. Actional Corporation. Año 2006.

[SOD-P&H] Service-oriented design and development methodology. M. P. Papazoglou Y W. J. Van Den Heuvel. International Journal of Web Engineering and Technology. 2006.

[SOMA-PPT] Services Oriented Architecture (SOA). Concepts. BabakHosseinzadeh. Copyright IBM Corporation. Año 2006.

[SUN-RQ] SOA RQ Methodology. A Pragmatic Approach. WWW. sun.com/soa. Sun Microsystems. Año 2006.

[TODO-AGIL] An Introduction to Agile Methods. Steve Hayes de Khatovar Technology y Martin Andrews de Object Consulting.

[UML-BOOCH] UML Users Guide. Grady Booch. Addison-Wesley. Año 1998.

[XML/EDI] Un Nuevo marco para el intercambio electónico de datos: XML/EDI. F. Araque Cuenca, M. V. Hurtado Torres y M.M. Abad Grau. Dpto. LSI - Facultad de C.C. E.E. y E.E. Universidad de Granada. Año 2004.

[XP-AGILEMETH] An Introduction to Agile Methods. Steve Hayes. Martin Andrews.

[XP-BRCONS] BR-Consulting. La Mejor Metodología 'Liviana' de Desarrollo de Software: eXtreme Programming. Javier Blanqué. Año 2000.

[XP-CASEST] eXtreme Programming applied: a case in the private banking domain. Piergiuliano Bossi. Quinary SpA.Munich 2003.

[XP-CASEST2] Exploring Extreme Programming in Context: An Industrial Case Study. Lucas Layman, Laurie Williams y Lynn Cunningham. North Carolina State University. , Department of Computer Science. Clarke College. Año 2004.

[XPEVAL-FR] Extreme Programming Evaluation Framework for Object-Oriented Languages. L. Williams, W. Krebs, and L. Layman. North Carolina State University Department of Computer Science. 06/04/2004.

[XP-INSTALL] Extreme Programming Installed. *Ron E*. Jeffries, Ann Anderson, Ann Anderson, Chet Hendrickson, Chet Hendrickson. Pearson Education. Octubre de 2000.

[XP-INTRO] Introducción a Extreme Programming. Gerardo Fernández Escribano. 9/12/2002.

[XP-REFACT] Extreme Programming refactored. The Case against XP. Matt Stephens, Doug Rosenberg. Año 2003.

[XP-REFACTORED] Extreme Programming Re-Factored: The case against XP. M. Stephens y D. Rosenberg. Julio de 2003.

[XQ-BECK] Extreme Programming Explained. Embrace Change. Kent Beck. Pearson Education. Año 1999.