UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

PROPUESTA DE UN MODELO DE PRUEBA PARA UNA ARQUITECTURA ORIENTADA A SERVICIOS

Trabajo de Diploma

Presentado para optar por el título de Ingeniero en Ciencias Informáticas.

Autor: Elizabeth Ochoa Luis
Ramón Rivero Torres

Tutor: Ing. Manuel A. Gil Martín
Ing. Leevan Abon Cepeda

Ciudad de la Habana, Cuba. Enero de 2009.

"*Año del 50 aniversario del triunfo de la Revolución*"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores del trabajo titulado:

PROPUESTA DE UN MODELO DE PRUEBA PARA UNA ARQUITECTURA SOA

Y autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____	_____
Elizabeth Ochoa Luis	Ramón Rivero Torres
Autor	Autor
_____	_____
Manuel A. Gil Martín	Leevan Abon Cepeda
Tutor	Tutor

AGRADECIMIENTOS.

Agradecemos a nuestras familias, amigos, a Leevan y a Manuel, y a todas las personas que de una forma u otra nos ayudaron con la realización de esta investigación.

DEDICATORIA.

Eliza:

Dedico esta tesis:

***A mis padres**, por enseñarme a ser la persona que soy, por darme todo el apoyo, el amor, el cariño y toda la guía que me dieron en estos 22 años, lo que escriba no basta para agradecerles. Papito gracias por tus exigencias y educación. Mamita gracias por ser mi mejor amiga y todos tus mimos.*

***A mi tía Diana** por apoyarme todos estos 5 años, eres mi tía preferida aunque no lo creas.*

***A Rey** por acompañarme en estos 4 años, por ser el hombre especial que es, por todo su amor y apoyo.*

***A Ramón**, eres el mejor compañero de tesis que he tenido, gracias por tu paciencia y por tu apoyo.*

***A mis tutores**, Leevan y Manuel, sin ellos no hubiera sido posible esta tesis, son dos personas excepcionales tanto como tutores como amigos, gracias por estar a mi lado en este momento tan decisivo en mi vida.*

A todos mis amigos...

Ramón:

Dedico esta tesis:

A mi madre, a mi familia y amigos...

RESUMEN.

El desarrollo de software implica una serie de actividades de producción en las que las posibilidades de que aparezca la fiabilidad humana son comunes. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software ha de ir acompañado de una actividad que garantice la calidad. El Centro de Consultoría Tecnológica y de Integración de Sistemas en la Universidad de las Ciencias Informáticas, dado su carácter productivo, se ha insertado al igual que otras grandes empresas en el mundo al intenso proceso que existe hoy en día en Internet enfocado a los servicios Web y al estilo arquitectónico SOA, por lo que se hace necesario asegurarse de que las aplicaciones que se obtengan como resultado en esta entidad sean eficientes, seguras e íntegras.

En la UCI no existe un modelo de pruebas para este tipo de arquitectura, es por ello que surge la idea de investigar y proponer un modelo capaz de guiar todo el proceso de desarrollo del software con el fin de ser usado por cualquier proyecto productivo que siga una línea SOA/BPM. Este modelo tiene como finalidad proponer principalmente la metodología a usar durante el desarrollo de las pruebas, los entregables que recogerán todos los datos de cada fase y los roles involucrados.

PALABRAS CLAVE

Calidad, pruebas, modelo, servicios, SOA

ÍNDICE

INTRODUCCIÓN.

Hace poco más de una década, la Arquitectura Orientada a Servicios (SOA, por sus siglas en inglés), comenzaba a escucharse por primera vez. Sin embargo, aproximadamente en el 2003, el concepto de Arquitectura Orientada a Servicios comenzó a expandirse y a encontrar sentido en un mundo en el que las aplicaciones de negocios cobran cada vez más relevancia y donde la rapidez de su puesta en marcha determina, muchas veces, el éxito o fracaso de la estrategia comercial de las compañías. SOA no es un concepto nuevo, los ingenieros de software entendieron sus principios a mediados de los 80 cuando llegaron al mercado la computación distribuida y las llamadas a procedimientos remotos. Años después, en 1996 Gartner definió:

"SOA es una arquitectura de software que comienza con una definición de interfaz y construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces. Sería mejor llamada "arquitectura orientada a interfaces". SOA es una relación de servicios y consumidores de servicios, ambos suficientemente amplios para representar una función de negocios completa". (1)

A pesar que Gartner dejó sentadas las bases de lo que sería SOA, no es hasta el 2003 que esta arquitectura entra por completo en el mundo de las TI (Tecnologías de la Información) empresariales, a través de los servicios web, surgiendo como la mejor manera de afrontar el desafío de hacer más con menos recursos, convirtiéndose en la nueva filosofía que permite soportar los requerimientos tecnológicos actuales y futuros de las empresas que estén decididas a prosperar en la nueva economía y proporcionando un puente entre TI y procesos de negocio ayudando a incrementar la flexibilidad de los mismos y la capacidad de las empresas para cumplir más rápida, fácil y económicamente las metas de negocio. Con SOA la necesidad de realizar pruebas sigue existiendo, sin embargo muchos servicios SOA pueden no tener una interfaz de usuario, lo cual es uno de los nuevos retos para la realización de pruebas al sistema.

El Centro de Consultoría Tecnológica y de Integración de Sistemas en la Universidad de las Ciencias Informáticas, dado su carácter productivo, se ha insertado al igual que otras grandes empresas en el mundo al intenso proceso que existe actualmente en Internet enfocado a los servicios Web y al estilo arquitectónico SOA. Producto de la novedad y del desconocimiento existente acerca del tema, realizar el proceso de pruebas de calidad resulta complejo y no prestarle la debida atención puede originar que se

cometan errores durante el desarrollo de las aplicaciones que pueden derivar en importantes pérdidas económicas.

Debido a que los procesos serios de desarrollo de software, en la mayoría de los casos tienden a ser caóticos, es necesario involucrar procesos de aseguramiento de la calidad, para que se puedan cumplir de manera correcta los requerimientos que el cliente necesita. Por otro lado, el costo que implica reparar un defecto que es descubierto en etapas avanzadas del desarrollo de software, tal como la implementación, es muy alto, hablando en términos del presupuesto del proyecto, como también en el cronograma.

No probar la arquitectura SOA implica que los servicios no se hagan con la calidad necesaria, puede que no se logren conectar bien a las herramientas de procesos, arruinando toda la inversión en la iniciativa SOA de la empresa, además que los clientes se encontrarán insatisfechos con el resultado final del sistema.

Ante el análisis de la **situación problémica** expuesta, se plantea como **problema científico** ¿Cómo guiar las pruebas de calidad en proyectos basados en una arquitectura orientada a servicios?

El **objeto de estudio** de esta investigación son las pruebas de calidad desarrolladas por los proyectos productivos en la Universidad de las Ciencias Informáticas y se define como **campo de acción** las pruebas de calidad de una Arquitectura Orientada a Servicios en el Centro de Consultoría Tecnológica y de Integración de Sistemas de la Universidad de las Ciencias Informáticas.

Para darle solución al problema planteado se precisó como **objetivo general** definir un modelo de pruebas para su utilización en proyectos desarrollados en una Arquitectura Orientada Servicios.

Durante la investigación se defiende la idea, que si se aplican pruebas de acuerdo a un modelo establecido, sobre proyectos desarrollados en una Arquitectura Orientada a Servicios, se reducen los posibles errores, y se aumenta potencialmente la calidad de los mismos.

Los **objetivos específicos** que se derivan del objetivo general son:

1. Realizar una valoración crítica de la situación actual de los modelos de pruebas existentes en el mundo en una arquitectura orientada a servicios.
2. Proponer un modelo de prueba para una Arquitectura SOA.
3. Definir los artefactos a utilizar en el modelo de prueba propuesto.

4. Proponer indicadores para la selección de herramientas de prueba.
5. Validar la propuesta del modelo de pruebas.

Las **tareas** a desarrollar para cumplir los objetivos de la investigación son las siguientes:

- Estudiar los conceptos relacionados con la gestión de la calidad.
- Estudiar el Estado del Arte de los modelos de pruebas existentes en el mundo.
- Estudiar los fundamentos de una Arquitectura Orientada a Servicios.
- Analizar los antecedentes y las tendencias actuales de la Arquitectura Orientada a Servicios en el mundo.
- Estudiar las diferencias y similitudes entre una arquitectura tradicional y la orientada a servicios.
- Estudiar los artefactos generados por las pruebas realizadas en una Arquitectura SOA.
- Estudiar las herramientas existentes para la realización de pruebas en una arquitectura orientada a servicios.
- Estudiar el método Delphi para validar la solución propuesta mediante el uso de expertos.

Estrategia de la Investigación

De acuerdo a los conocimientos existentes respecto al problema planteado, los tipos de estrategias que permiten estructurar y guiar nuestra investigación son: una estrategia exploratoria, ya que la fuente de información acerca del tema es poca y la mediana experiencia en dicho tema hace que no se tenga una idea clara del asunto en cuestión, y una estrategia descriptiva porque mediante la investigación se quiere reflejar la idea central del problema en cuestión y así comprender el valor científico de los resultados obtenidos.

Los **métodos científicos** utilizados son:

- **Métodos teóricos:**
    - **Método Analítico - Sintético**: Durante todo el proceso investigativo se hizo un estudio a profundidad de toda la bibliografía relacionada con el tema y a partir del análisis realizado se seleccionó una síntesis de lo estudiado.

- **Método Histórico - Lógico**: Para la realización de la investigación se hizo necesario estudiar la evolución del problema y la existencia de modelos de prueba similares al que se pretende elaborar.

- **Métodos empíricos:**
  - **Observación**: Consiste en la percepción planificada dirigida a un fin y relativamente prolongada de un hecho o fenómeno. Este método permitió percibir directamente cómo se hacen las pruebas de calidad en la práctica para así poder llevar esos conocimientos al tema propuesto.
  - **Encuesta**: Durante la realización de la investigación fue necesario realizar encuestas para la validación de la solución propuesta. A través de este método se conocerá el criterio de los expertos, en la clasificación de la solución en desarrollo que se podrá llevar a la práctica en caso de ser aceptada.

Resultados Esperados

Propuesta de un modelo de prueba para su utilización en proyectos desarrollados sobre una Arquitectura Orientada Servicios que permita al Centro de Consultoría Tecnológica y de Integración de Sistemas en la Universidad de las Ciencias Informáticas asegurar la calidad de las mismas.

Este trabajo consta de introducción, tres capítulos y conclusiones.

El Capítulo 1: **Fundamentación Teórica**, aborda los temas relacionados con la calidad de software, pruebas de calidad de software, conceptos relacionados con SOA y las diferencias que existen entre una arquitectura SOA y una arquitectura tradicional.

El Capítulo 2: **Solución Propuesta**, en este capítulo se realizará un estudio de pruebas realizadas en SOA y se definirá un modelo de prueba, los artefactos que se generan en dicho modelo y una propuesta de indicadores que debería cumplir cualquier herramienta de pruebas de calidad.

El Capítulo 3: **Validación de la Solución Propuesta**, en este capítulo se realizará la validación del modelo de prueba para una arquitectura SOA, con la ayuda del método Delphi.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción.

En este capítulo se detallan y fundamentan conceptos relacionados con los tipos de pruebas y las arquitecturas orientadas a servicios, luego se reflejan las diferencias entre una arquitectura SOA y una arquitectura tradicional, dando paso a la solución propuesta de nuestra investigación.

1.2 Arquitectura Orientada a Servicios.

La Arquitectura Orientada a Servicios (SOA, por sus siglas en inglés) es un estilo arquitectónico empresarial distribuido para la construcción de aplicaciones de software, que utilizan los servicios disponibles en una red como la web, el mismo está basado en estándares en el que:

- Se separan formalmente los servicios de sus consumidores.
- Los proveedores del servicio publican un contrato que será la base para su consumo.
- Existe un acoplamiento débil entre proveedores y consumidores de servicios.

En muchas ocasiones se confunde SOA con una tecnología o producto software, y nada más lejos de la realidad. SOA es un concepto de diseño de arquitectura que intenta mezclar a las tecnologías de la información con el propio negocio de la organización (para que las soluciones que aporte sean lo más cercanas a los requisitos de negocio que se intenten cubrir, y dejen de ser soluciones departamentales que cubran o resuelvan solo parcialmente parte de las necesidades existentes sin tener una visión de la globalidad del proceso), conjuntamente sugiere la creación de servicios y funcionalidades de negocio que sean fácilmente reutilizables. Estos servicios deben ser flexibles, seguros y lo más importante de todo, con una arquitectura basada en estándares. (2)

Un objetivo fundamental de SOA, es que los componentes sean manejables, que se adapten fácilmente a los cambios. Los servicios deben ser flexibles, y los procesos de negocio implementados sobre SOA también, luego los cambios tecnológicos, o de negocio afectarán menos a una empresa, y específicamente afectarán en menor medida a los sistemas que utilizan dichos componentes. Esta flexibilidad permite salir en forma oportuna con nuevos productos, o soluciones.

La reutilización, y flexibilidad son los fundamentos de SOA, solo con ellos se consigue bajar los costos de mantención, se logra aprovechar de mejor forma los recursos implementados, mejorar la productividad y el time2market, es decir obtener el retorno de la inversión.

Lograr este tipo de componentes reutilizables y flexibles e implementar soluciones basadas en ellos, no es una tarea fácil, porque no basta solo con utilizar ciertas herramientas, sino además se apoya fuertemente en el cumplimiento de estándares y buenas prácticas y es por eso que se denomina “Arquitectura” (Orientada a Servicios). Se pueden desarrollar componentes sobre tecnología SOA, pero aun así no asegurar su reutilización, o flexibilidad. (3)

En una arquitectura SOA 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 pero no se reduce a la integración de estas dentro de una localización concreta, sino que va más allá, va a los procesos de las organizaciones, a la gobernabilidad, al uso de tecnología estándar, a la integración en entornos distribuidos.

Un típico error a la hora de analizar el concepto de SOA es pensar siempre en Servicios Web como estándar de definición de un servicio, pero esto no es así, ya que por norma general solamente entre el 20% y el 40% de los servicios usarán esta interfaz. La mayoría de los clientes empiezan a practicar con SOA creando un Servicio Web, muchas veces sobre lógica de negocio ya existente. Aunque los Servicios Web no necesariamente significan SOA y no todas las Arquitecturas Orientadas a Servicios están basadas en Servicios Web, la relación entre las dos tendencias es importante, y se potencian mutuamente; el interés por los Servicios Web lleva hacia SOA y las ventajas de la arquitectura SOA ayudan a que las iniciativas de Servicios Web tengan éxito.

1.2.1 Servicios Web.

Un servicio Web es una colección de protocolos y estándares empleados para intercambiar datos entre aplicaciones y sistemas. Las aplicaciones, escritas en diversos lenguajes de programación y ejecutándose en distintas plataformas pueden utilizar los servicios Web para intercambiar datos sobre una red de ordenadores como Internet de una forma similar a la comunicación entre procesos en un solo ordenador. (4)

Los Servicios Web son un mecanismo de comunicación distribuida que permiten que las aplicaciones compartan información y que además invoquen funciones de otras aplicaciones independientemente de cómo se hayan creado, cuál sea su sistema operativo o la plataforma en la que se ejecutan y cuáles sean los dispositivos utilizados para acceder a ellas. En los servicios Web, todos los datos que se intercambian se formatean con etiquetas XML.

1.2.2 XML.

XML, siglas en inglés de Extensible Markup Language (lenguaje de marcado extensible), es un metalenguaje extensible de etiquetas. Permite definir la gramática de lenguajes específicos, por lo tanto no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Este concepto no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. (5)

XML representa la arquitectura de capas base de la arquitectura SOA. Dentro de ella, establece el formato y la estructura de los mensajes que viajan a través de servicios. En otras palabras, no se puede realizar un movimiento dentro de una SOA sin su participación. Para que un documento XML sea bien formado debe seguir las reglas básicas de sintaxis. Para que sea válido, debe estar bien formado, hacer referencia a un DTD o un Esquema XML y ser conforme a la gramática definida en dicho esquema.

1.2.3 Esquemas.

XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así, una percepción del tipo de documento con un nivel alto de abstracción. Fue desarrollado por el World Wide Web Consortium (W3C) y alcanzó el nivel de recomendación en mayo del 2001.
En un esquema XML se definen cuatro tipos básicos de información, cada uno de los cuales proveen la clase de información que un usuario necesita saber para utilizar un servicio web de otra empresa.

Los cuatro tipos de información son: (6)

- Información del negocio: Describe la empresa, su nombre, los contactos, el tipo de empresa y los objetivos principales.
- Información del servicio: Informa sobre servicios web generalmente agrupados por procesos de negocio o categorías de servicios.
- Información del enlace (binding): Cada uno de ellos brinda una dirección física para hacer contacto con los servicios descriptos anteriormente.
- Información sobre las especificaciones del servicio: Brinda información sobre las especificaciones del servicio, por ejemplo, como tienen que ser los mensajes que el servicio espera y responde, etc.

Esta información es utilizada por los registros UDDI para publicar descripciones de servicios (WSDL) y permitir su descubrimiento.

1.2.4 UDDI.

UDDI, Universal Discovery Description and Integration, es un modelo de directorios para servicios web. Es una especificación para mantener directorios estandarizados de información acerca de los servicios web, sus capacidades, ubicación, y requerimientos en un formato reconocido universalmente. UDDI utiliza WSDL para describir las interfaces de los servicios web. Es un lugar en el cual podemos buscar cuáles son los servicios web disponibles, una especie de directorio en el cual podemos encontrar los servicios web publicados y así publicar los que desarrollemos (7). UDDI ofrece la funcionalidad necesaria para el descubrimiento y registro de los servicios ofrecidos por una determinada entidad.

Dos partes claramente diferenciadas:

- Directorio con todos los metadatos de los servicios web publicados, con una referencia a cada servicio WSDL de cada uno.
- Definiciones de los “port types” para buscar y manipular en esos directorios.

Estándares:

- White pages (Información general)
- Yellow pages (Categorías de servicios)
- Green pages (Reglas de negocio)

Mientras que la UDDI permite el manejo de los Servicios Web y la publicación de los mismos, existe una herramienta muy potente que proporciona una comunicación fiable entre los distintos recursos tecnológicos tales como aplicaciones, plataformas y servicios, que están distribuidos en múltiples sistemas por toda la empresa, el ESB.

1.2.5 ESB.

Un bus de servicios empresariales (ESB) es una tecnología o producto software que puede definirse como la Infraestructura que sirve como el backbone de las Arquitecturas Orientadas a Servicios (SOA). El mismo permite a una empresa, conectar, mediar, y controlar la interacción entre diversas aplicaciones y servicios a lo largo de entornos altamente distribuidos y heterogéneos. Normalmente un ESB debe proporcionar a una organización, por un lado, un sistema de mensajería robusto de alta disponibilidad y altamente escalable que permita garantizar la comunicación entre los distintos servicios y aplicaciones de la organización independientemente de su localización geográfica, y por otro lado, un mecanismo que permita fácilmente la definición de nuevos procesos o su posterior modificación orquestando los servicios existentes en la organización. (2)

Además debe proporcionar funcionalidades avanzadas como la de enrutamiento de datos dinámico basado en el contenido de estos. Resumiendo, ESB desde el punto de vista SOA es la herramienta que permite desarrollar procesos basado en servicios SOA. Existen otras tecnologías que guardan mucha relación con SOA, que son herramientas que permiten organizar actividades en un flujo. Bajo este concepto podemos tomar a BPM (Business Process Management). La diferencia con BPM es que el ESB no maneja las actividades humanas solo maneja actividades automatizadas.

1.2.6 BPM.

Business Process Management (BPM) es un conjunto de métodos, herramientas y tecnologías utilizados para diseñar, representar, analizar y controlar procesos de negocio operacionales, es un enfoque centrado en los procesos para mejorar el rendimiento que combina las tecnologías de la información con metodologías de proceso y gobierno y una colaboración entre personas de negocio y tecnólogos para fomentar procesos de negocio efectivos, ágiles y transparentes. (8)

Como su nombre sugiere, BPM se enfoca en la gestión de los procesos del negocio. La automatización de los procesos reduce errores, asegurando que los mismos se comporten siempre de la misma manera y dando elementos que permitan visualizar el estado de los mismos. La administración de los procesos permite asegurar que los mismos se ejecuten eficientemente y la obtención de información que luego puede ser usada para mejorarlos. Es a través de la información que se obtiene de la ejecución diaria de los procesos, que se puede identificar posibles ineficiencias en los mismos y actuar sobre las mismas para optimizarlos.

Para soportar esta estrategia es necesario contar con un conjunto de herramientas que den el soporte necesario para cumplir con el ciclo de vida de BPM. Este conjunto de herramientas son llamadas Business Process Management System y con ellas se construyen aplicaciones BPM.

Beneficios de BPM:

- **Automatización**: mayor productividad, coherencia, reducción de errores, mayor satisfacción del cliente y conformidad.
- **Agilidad**: tiempos más rápidos de respuesta a los problemas, tiempos más rápidos para desarrollar soluciones y para responder de forma inmediata.
- **Flexibilidad**: combinación de escala, alcance y capacidad de los sistemas de información tradicionales con la agilidad, flexibilidad e innovación de las modernas tecnologías como Web 2.0 y mejora de una plataforma de información con las herramientas y técnicas de CPI, indicadores de desempeño (Balanced Scorecards), metodología, gobierno, entornos de trabajo y metadatos.

- **Visibilidad**: realizar el seguimiento de transacciones empresariales individuales (incluso en tiempo real) por todo el proceso, penetrando en los subprocesos, acercándose a los procesos principales, y viendo el proceso desde la perspectiva de un rol en particular.
- **Colaboración**: alineamiento y participación, especialmente entre TI y el negocio.
- **Gobierno:** un modelo fuerte de control y cambio de la gestión que crea confianza en los clientes, socios, proveedores, reguladores y accionistas. BPM garantiza el seguimiento de las políticas de utilización y reutilización, y proporciona supervisión de las tareas y del flujo de trabajo.

Si a través del modelado de las actividades y procesos puede lograrse un mejor entendimiento del negocio y muchas veces esto presenta la oportunidad de mejorarlos, con un modelo de madurez para SOA se describen las capacidades de una arquitectura SOA efectiva y en particular se brinda una visión de donde se debe mejorar.

1.2.7 Modelo de Madurez SOA.

Permite medir el estado actual de una empresa con respecto a la utilización de SOA: Proyectos, Arquitectura, Infraestructura, Gobierno y define una ruta de evolución. El modelo está compuesto por 5 niveles que tienen una relación directa con los niveles de CMM donde el principal objetivo es medir la calidad del desarrollo del software. (9)

Fig. 1.1. Modelo de Madurez SOA.

Nivel 1: Servicios SOA iniciales.

Propósito: Introducir funcionalidad y nueva tecnología.

Alcance: Proyectos piloto, investigación y experimentación, proyectos pequeños.

Nivel 2: Servicios SOA arquitecturados.

Propósito: Reducción de costos y control de TI.

Alcance: Múltiples aplicaciones integradas dentro de la organización.

Nivel 3(a): Servicios de negocio SOA.

Propósito: Agilidad del negocio.

Alcance: Procesos de negocio a través de empresas o unidades de negocio distintas.

Nivel 3(b): Servicios de colaboración SOA.

Propósito: Colaboración con asociados de negocio y mercado en general.

Alcance: Abarca múltiples organizaciones internas o internas y externas.

Nivel 4: Servicios SOA de métricas del negocio.

Propósito: Encontrar medidas de desempeño del negocio.

Alcance: Unidades de negocio y/o a través de la empresa.

Nivel 5: Servicios de Negocio Optimizados SOA.

Propósito: Optimización del negocio, reaccionar y responder automáticamente.
Alcance: Unidades de negocio y/o a través de la empresa.

1.2.8 QoS.

La Calidad de Servicio (QoS, Quality of Service) es el efecto colectivo del desempeño de un servicio, el cual determina el grado de satisfacción a la aplicación de un usuario (10), garantiza la transmisión de cierta cantidad de datos en un tiempo dado y es la capacidad de dar un buen servicio. QoS para aplicaciones que deben comunicarse en tiempo real, es el conjunto de características cualitativas y cuantitativas de un sistema multimedia distribuido, que son necesarias para lograr la funcionalidad requerida en una aplicación (11). En este sentido, determina la usabilidad y utilidad del servicio, influenciando en la popularidad del mismo.

Con la extensa propagación de los servicios Web, QoS se ha convertido en un factor significante para determinar el éxito que pueda alcanzar un proveedor de servicio sobre otros. Cubre un rango de técnicas que armoniza las necesidades de quien requiere el servicio con quién lo provee, basándose en los recursos de red disponibles. A través de QoS se puede hacer referencia a las propiedades no funcionales del servicio Web, tales como rendimiento, fiabilidad, disponibilidad y seguridad, las cuales representan los requerimientos de calidad necesarios para alcanzar la funcionalidad esperada del mismo. Estos servicios Web deben tener un modelo de acuerdo de nivel de servicios (SLA).

1.2.9 SLA.

EL modelo de Acuerdo de Nivel de Servicios (Service Level Agreement, SLA) consiste en un contrato en el que se estipulan los niveles de un servicio en función de una serie de parámetros objetivos, establecidos de mutuo acuerdo entre ambas partes, así, refleja contractualmente el nivel operativo de funcionamiento, penalizaciones por caída de servicio y limitación de responsabilidad por no servicio. Este modelo no ha de estar relacionado necesariamente con la contratación de servicios a terceras partes, sino que puede implantarse a nivel interno, transformando una determinada unidad de negocio en centro de servicios que provea a la propia compañía (12). Para implantar con éxito un SLA han de tenerse en cuenta una serie de factores claves, de los que va a depender en gran medida la obtención de los resultados deseados:

Aspectos críticos:

Los aspectos más críticos, son la definición de procedimientos estándares y los mecanismos de evaluación y seguimiento.

En la implantación de un SLA se deben seguir una serie de puntos:

- Definición de objetivos: mejora de la eficacia, reducción de costos, formalización de la relación.
- Identificación de expectativas: qué es lo que espera la organización de este acuerdo.
- Adecuada planificación temporal.
- Optimización/rediseño de procesos: revisar los procesos si el SLA no asegura ningún cambio o como mínimo formalizarlos.

Errores más frecuentes en la implantación:

- Definición de niveles de servicio inalcanzables.
- Regulación excesiva.
- Error en la definición de prioridades.
- Complejidad técnica.
- Irrelevancia (si un SLA no tiene ningún efecto sobre el cliente, el objetivo no tiene sentido).

Todos los servicios Web además de tener un SLA deben tener medidores de rendimiento, como son los KPI.

1.2.10 KPI.

Indicadores clave de rendimiento (Key Performance Indicators KPI), miden el nivel del desempeño de un proceso, enfocándose en el cómo e indicando como de buenos son los procesos, de forma que se pueda alcanzar el objetivo fijado. (13)

Es necesario para una organización al menos identificar sus KPI. El ambiente clave para identificar los KPI son:

- Tener un proceso del negocio pre-definido.
- Tener claros los requerimientos de las metas y desempeño del negocio.
- Tener mediciones cualitativas y cuantitativas de los resultados y compararlos con el conjunto de metas.
- Investigar la variabilidad y debilidades del proceso o los recursos para lograr las metas en el corto plazo.

Los KPI difieren dependiente de la naturaleza de una organización y de su estrategia. Ayudan a la organización a medir el progreso hacia el logro de las metas de las organizaciones, especialmente hacia la dificultad de cuantificar el conocimiento basado en actividades. Quien se encarga de medir el rendimiento KPI es el BAM.

1.2.11 BAM.

BAM (Business Activity Monitoring) es una estrategia de negocios que permite integrar diferentes fuentes de información, para generar alarmas en tiempo real dirigidas a personas que toman las decisiones adecuadas sobre procesos claves del negocio con la información suficiente y relevante para tomar acción sobre ellos. BAM le da la capacidad de acceder a sus procesos críticos de negocio y mediante indicadores controlar la eficiencia y la velocidad de sus operaciones. (14)

Beneficios obtenidos con las soluciones BAM

El uso de soluciones BAM en una entidad permite:

- Obtener información en tiempo real acerca del desarrollo de la actividad del negocio, que posibilita la rápida detección de problemas, permitiendo la identificación de su causa y el análisis de su impacto.
- Automatización de los procesos de control manual, dando lugar a la disminución de los costes/riesgos operacionales y el aumento de la eficiencia.
- Monitorización de los procesos/actividades empresariales en relación a los compromisos adquiridos en el negocio y expectativas del cliente.
- Aumento de la satisfacción y fidelidad del cliente a través de la gestión pro-activa.

- Permite la integración de una variedad de sistemas.

En la Arquitectura Orientada a Servicios debe estar presente un fuerte proceso de calidad desde que se inicie el desarrollo de la misma con el diagnóstico inicial, puesto que desde los primeros pasos en cualquier tipo de desarrollo de software, el factor calidad marca el correcto desempeño por parte del equipo de desarrollo y este, a su vez, debe lograr que dicho factor trascienda a lo largo de todo el ciclo de vida del proyecto trayendo consigo una mejor implantación del sistema.

1.3 Calidad Del Software.

Todas las metodologías y herramientas tienen un único fin, producir software de gran calidad. La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad. La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario. (15). La misma es medible y varía de un sistema a otro o de un programa a otro. Puede medirse después de elaborado el producto, pero esto puede resultar muy costoso si se detectan problemas derivados de imperfecciones en el diseño, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad como su control durante todas las etapas del ciclo de vida del software. Los Factores de calidad representan la calidad desde el punto de vista del usuario y son las características que componen la calidad, también son denominados Atributos de Calidad Externos.

1.3.1 Factores de Calidad.

A pesar del avance en el desarrollo de software y las tecnologías, con el paso de los años los atributos que proporcionan una indicación de la calidad del software siguen siendo los mismos. La norma ISO 9126 es un ejemplo de ello, muchas características y sub-características definidas en la misma hacen referencia a la operación, transición y revisión del software y señalan entre otras cosas la necesidad de lograr que el software opere correctamente y con el grado de exactitud requerido, que los usuarios sean capaces de entenderlo y usarlo, es decir que sea amigable con quienes interactúen con él, que sea capaz de responder correctamente ante fallos o cambios del entorno y que proporcione una ejecución o desempeño apropiado, teniendo en cuenta los recursos utilizados.

Los factores que determinan la calidad del software se clasifican en tres grupos: (16)

- **Operaciones del producto: características operativas**
  - **Corrección**: Grado en que un programa satisface sus especificación y logra los objetivos marcados por el usuario. (¿Hace lo que se le pide?).
  - **Fiabilidad**: Grado en que se puede esperar que un programa lleve a cabo las funciones esperadas con la precisión requerida. (¿Lo hace de forma fiable todo el tiempo?).
  - **Eficiencia**: Cantidad de recursos de computadoras y de código requeridos por el programa para realizar sus funciones con los tiempos de respuesta adecuados. (¿Qué recursos hardware y software necesito?).
  - **Integridad**: Grado en que puede controlarse el acceso al software o a los datos por usuarios no autorizados. (¿Puedo controlar su uso?).
  - **Facilidad de uso**: Esfuerzo necesario para aprender, utilizar, preparar las entradas e interpretar las salidas de un programa. (¿Es fácil y cómodo de manejar?).

- **Revisión del producto:** capacidad para soportar cambios.
  - **Facilidad de mantenimiento**: Esfuerzo requerido para localizar y arreglar un error en un programa. (¿Puedo localizar los fallos?).
  - **Flexibilidad**: Esfuerzo requerido para modificar un programa. (¿Puedo añadir nuevas opciones?).
  - **Facilidad de prueba**: Esfuerzo requerido para probar un programa de forma que se asegure que realiza la función requerida. (¿Puedo probar todas las opciones?).

- **Transición del producto:** adaptabilidad a nuevos entornos.
  - **Portabilidad**: Esfuerzo requerido para transferir un programa desde un entorno HW y/o SW a otro. (¿Podré usarlo en otra máquina?).
  - **Reusabilidad**: Grado en que un programa o componente SW se puede reutilizar en otras aplicaciones. (¿Podré utilizar alguna parte del software en otra aplicación?).
  - **Interoperabilidad**: Esfuerzo requerido para acoplar un sistema con otras aplicaciones o sistemas. (¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?).

Producto a la necesidad de tener una alta calidad del software se asume que es más rentable prevenir los fallos de calidad que corregirlos o lamentarlos, y se incorpora el concepto de la "prevención" a la Gestión de la Calidad, que se desarrolla sobre esta nueva idea en las empresas industriales, bajo la denominación de Aseguramiento de la Calidad.

1.3.2 Aseguramiento de la Calidad.

El aseguramiento de calidad del software es el conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto (software) satisfará los requisitos dados de calidad (17). El aseguramiento de calidad del software se diseña para cada aplicación antes de comenzar a desarrollarla y no después.

Algunos autores prefieren decir *garantía de calidad* en vez de aseguramiento.

- Garantía, puede confundir con garantía de productos.
- Aseguramiento pretende dar confianza en que el producto tiene calidad.

El aseguramiento de calidad del software está presente en:

- Métodos y herramientas de análisis, diseño, programación y prueba.
- Inspecciones técnicas formales en todos los pasos del proceso de desarrollo del software.
- Estrategias de prueba multiescala.
- Control de la documentación del software y de los cambios realizados.
- Procedimientos para ajustarse a los estándares (y dejar claro cuando se está fuera de ellos).
- Mecanismos de medida (métricas).
- Registro de auditorías y realización de informes.

Hoy en día las compañías que producen software deben buscar continuamente alternativas que les permitan mejorar su performance y calidad de productos para poder seguir compitiendo en un escenario cada vez más globalizado y agresivo. A nivel mundial, la calidad del proceso utilizado para desarrollar un determinado producto impacta fuertemente en la calidad final. En consecuencia, el mejoramiento de los procesos de desarrollo forma parte de la estrategia competitiva. El SEI (Software Engineering Institute) propone un modelo de clasificación y mejora de los procesos empleados por las organizaciones de software denominado CMMI (Modelo de Madurez de Capacidad Integrado).

1.3.3 CMMI.

CMMI (Capability Maturity Model Integration / Modelo de Madurez de Capacidad Integrado) es un proceso de mejora que proporciona a las organizaciones los elementos esenciales de procesos eficaces. Se puede utilizar para guiar la mejora del proceso a través de un proyecto, una división, o toda una organización. CMMI ayuda a integrar funciones tradicionalmente separadas de organización, establecer objetivos de mejora de procesos y las prioridades, servir de orientación para los procesos de calidad y proporcionar un punto de referencia para evaluar los procesos actuales. (18)

Evaluar la calidad a través de las normas CMMI no significa exactamente obtener una certificación, sino una evaluación satisfactoria mediante la aplicación del modelo. De esta manera se puede determinar si, de acuerdo al modelo adoptado, una compañía responde en lo que a metodologías, estándares y procesos se refiere. Con la evaluación de calidad una empresa busca diferenciarse, dar un valor agregado.

Los siguientes son algunos de los beneficios y las razones de negocio para la aplicación de mejora de procesos:

- La calidad de un sistema es altamente influenciado por la calidad del proceso utilizado para adquirir, desarrollar y mantener la misma.
- Proceso de mejora de productos y aumento de la calidad de los servicios que las organizaciones aplican para lograr sus objetivos empresariales.
- Proceso de mejora de los objetivos que están alineados con los objetivos empresariales.

Beneficios de aplicar CMMI:

- Vincular más explícitamente las actividades de ingeniería y gestión a sus objetivos de negocio.
- Ampliar el alcance y la visibilidad en el ciclo de vida del producto y las actividades de ingeniería para garantizar que el producto o servicio cumple las expectativas de los clientes.
- Incorporar la experiencia adquirida en otros ámbitos de las mejores prácticas (por ejemplo, la medición, gestión de riesgos y gestión de proveedores).
- Aplicar de forma más robusta las prácticas de alta madurez.
- Abordar otras funciones de organización fundamental para sus productos y servicios.

- Cumplir cabalmente con las normas ISO.

Los 6 niveles definidos en CMMI para medir la capacidad de los procesos son:

Nivel 0: Incompleto: El proceso no se realiza, o no se consiguen sus objetivos.
Nivel 1: Ejecutado: El proceso se ejecuta y se logra su objetivo.
Nivel 2: Gestionado: Además de ejecutarse, el proceso se planifica, se revisa y se evalúa para comprobar que cumple los requisitos.
Nivel 3: Definido: Además de ser un proceso gestionado se ajusta a la política de procesos que existe en la organización, alineada con las directivas de la empresa.
Nivel 4: Cuantitativamente gestionado: Además de ser un proceso definido se controla utilizando técnicas cuantitativas.
Nivel 5: Optimizante: Además de ser un proceso cuantitativamente gestionado, de forma sistemática se revisa y modifica o cambia para adaptarlo a los objetivos del negocio. Mejora continua.

Los niveles de madurez son una plataforma evolutiva bien definida destinada a lograr un proceso de software maduro. Cada nivel de madurez proporciona una capa en los cimientos para un proceso de mejora continua. Entonces, cada nivel comprende un conjunto de objetivos que, una vez alcanzados, estabilizan un componente importante del proceso de software. Al alcanzar cada nivel del marco de madurez se establece un componente diferente en el proceso de software, resultando en un incremento en la capacidad de proceso de la organización.

El desarrollo del software implica una serie de actividades de producción en las que las posibilidades de que aparezca la fiabilidad humana son comunes. Los errores pueden empezar a darse desde el primer momento del proceso en el que los objetivos pueden estar especificados de forma errónea e imperfecta; así en los posteriores pasos del diseño y desarrollo. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo del software ha de ir acompañado de una actividad que garantice la calidad.

1.4 Pruebas de Calidad de Software.

Definitivamente la prueba es una actividad fundamental en muchos procesos de desarrollo, incluyendo el del software. De manera general, se puede decir que la prueba de software permite al desarrollador

determinar si el producto generado satisface las especificaciones establecidas. Así mismo, una prueba de software permite detectar la presencia de errores que pudieran generar salidas o comportamientos inapropiados durante su ejecución.

De acuerdo a la IEEE [IEEE90] el concepto de prueba se define como, una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente (15). La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación.

Existen varias normas que sirven como objetivos de las pruebas:

- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Los objetivos anteriores, suponen un cambio importante de punto de vista ya que la idea normal es que una prueba tiene éxito si no descubre errores. El objetivo de la prueba es: “diseñar pruebas que saquen a la luz diferentes clases de errores con la menor cantidad de tiempo y espacio”.

Enfoques de Prueba. (19)

- Enfoque estructural o de caja blanca
- Enfoque funcional o de caja negra

1.4.1 Enfoque estructural o de caja blanca.

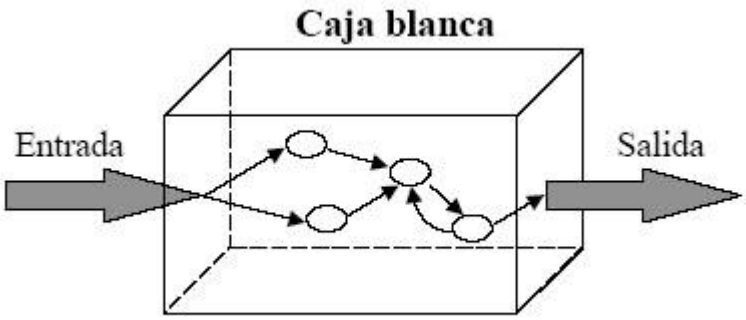

Fig. 1.2. Enfoque de diseño de pruebas de caja blanca

De acuerdo con Presman, la prueba de caja blanca denominada a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba.

Mediante este método el ingeniero de software puede obtener casos de prueba que:

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo.
- Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Existen dos términos fundamentales que describen la forma de realizar pruebas:

- Pruebas Estáticas
- Pruebas Dinámicas

Estos términos permiten definir dos clases de pruebas de caja blanca:

- **Pruebas estáticas de caja blanca:** Es el proceso que cuidadosamente y metódicamente revisa el diseño del software, la arquitectura o el código para encontrar defectos sin necesidad de ejecutar el código. Esto algunas veces se refiere a un análisis estructural.
- **Pruebas dinámicas de caja blanca:** En estas pruebas se revisa dentro de la caja, se examina el código y se observa éste mientras se ejecuta. La prueba de caja blanca dinámica, en resumidas palabras, utiliza la información que se obtiene al observar qué hace el código, cómo trabaja, para así determinar qué probar, qué no probar y cómo aproximarse a las pruebas.

1.4.2 Métodos de prueba de caja blanca.

Algunos de los métodos empleados en las pruebas de caja blanca son los siguientes:

- **Prueba del camino básico:** Es una técnica propuesta inicialmente por Tom McCabe, la cual le permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizarán que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

  Algunos elementos y conceptos utilizados alrededor de éste método son los siguientes:

  - Grafo de flujo o grafo del programa: Representa el flujo de control lógico de un programa y se utiliza para trazar más fácilmente los caminos de éste. (Cada nodo representa una o más sentencias procedimentales y cada arista representa el flujo de control).
  - Complejidad ciclomática: Es una métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto de las pruebas, el cálculo de la complejidad ciclomática representa el número de caminos independientes del conjunto básico de un programa. Esta medida ofrece al probador de software un límite superior para el número de pruebas que debe realizar para garantizar que se ejecutan por lo menos una vez cada sentencia.

- Camino independiente: Cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.

- **Prueba de la estructura de control:** Dentro de éste tipo de prueba se contempla el método del camino básico mencionado anteriormente pero además existen otras pruebas asociadas que permiten ampliar la cobertura de la prueba y mejorar su calidad. Estas son:

  - Prueba de condición: Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. Algunos conceptos empleados alrededor de esta prueba son los siguientes:
    - *Condición simple:* es una variable lógica o una expresión relacional (E1 < operador - relacional > E2).
    - *Condición compuesta:* está formada por dos o más condiciones simples, operadores lógicos y paréntesis.

  En general los tipos de errores que se buscan en una prueba de condición, son los siguientes:

  - Error en operador lógico (existencia de operadores lógicos incorrectos, desaparecidos, sobrantes).
  - Error en variable lógica.
  - Error en paréntesis lógico.
  - Error en operador relacional.
  - Error en expresión aritmética.

- **Prueba del flujo de datos:** Selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **Prueba de bucles:** Es una técnica que se centra exclusivamente en la validez de las construcciones de bucles (bucles simples, anidados, concatenados y no estructurados).

  Bucles simples:

  Se les aplica el siguiente conjunto de pruebas:
  - Pasar por alto totalmente el bucle.
  - Pasar una sola vez por el bucle.

- Pasar dos veces por el bucle.
- Hacer m pasos por el bucle con m < n (donde n es el número máximo de pasos permitidos por el bucle).
- Hacer n - 1, n y n + 1 pasos por el bucle.

Bucles anidados:

Si se empleara el mismo enfoque de prueba de bucles simples a los bucles anidados, el número de pruebas aumentaría considerablemente por lo que es recomendable emplear el siguiente enfoque:

- Comenzar por el bucle más interior. Establecer o configurar los demás bucles con sus valores mínimos.
- Llevar a cabo las pruebas de bucles simples para el bucle más interior, mientras se mantienen los parámetros de iteración de los bucles externos en sus valores mínimos. Añadir otras pruebas para valores fuera de rango o excluidos.
- Progresar hacia fuera, llevando a cabo pruebas para el siguiente bucle, pero manteniendo todos los bucles externos en sus valores mínimos y los demás bucles anidados en sus valores típicos.
- Continuar hasta que se hayan probado todos los bucles.

Bucles concatenados:

Estos bucles se pueden probar utilizando el enfoque de bucles simples, siempre y cuando cada uno de los bucles sea independiente del resto de lo contrario se debe emplear el enfoque de bucles anidados.

Bucles no estructurados:

Siempre que sea posible estos bucles deben rediseñarse.

1.4.3 Enfoque funcional o de caja negra.

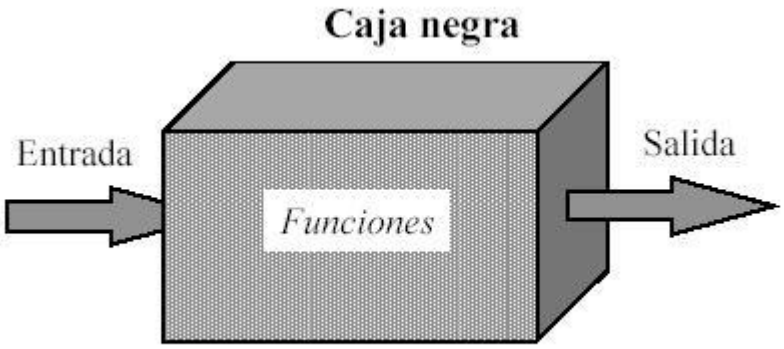

Fig. 1.3. Enfoque de diseño de pruebas de caja negra.

Las pruebas de caja negra se centran en los requisitos funcionales del software. Es decir, la prueba de caja negra permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Se trata de un enfoque que intenta descubrir diferentes tipos de errores que no se encuentran con los métodos de caja blanca.

La prueba de caja negra, intentan encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

La prueba funcional o de caja negra se centra en el estudio de la especificación del software, del análisis de las funciones que debe realizar, de las entradas y de las salidas. Existen dos términos fundamentales que describen la forma de realizar pruebas:

- Pruebas Estáticas
- Pruebas Dinámicas

Estos dos términos permiten definir dos clases de pruebas de caja negra:

- **Pruebas de caja negra estáticas:** Probando la especificación: la especificación es un documento, no es el programa ejecutándose; esto es lo que se considera estático. Entonces se puede tomar el documento, hacer las pruebas estáticas de caja negra y examinar cuidadosamente esta para encontrar errores.
- **Pruebas de caja negra dinámicas:** Es probar el software sin tener un conocimiento de los detalles del código fuente. Es dinámica porque el programa se está ejecutando mientras se está probando. Y es caja negra porque se prueba sin tener un conocimiento exacto de cómo trabaja. Se ingresan entradas, se reciben salidas, y se comprueban resultados.

1.4.4 Métodos de prueba de caja negra.

Algunos de los métodos empleados en las pruebas de caja negra son los siguientes:

- **Métodos de prueba basados en grafos:** En este método se debe entender los objetos (objetos de datos, objetos de programa tales como módulos o colecciones de sentencias del lenguaje de programación) que se modelan en el software y las relaciones que conectan a estos objetos. Una vez que se ha llevado a cabo esto, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. En este método:

    - Se crea un grafo de objetos importantes y sus relaciones.
    - Se diseña una serie de pruebas que cubran el grafo de manera que se ejerciten todos los objetos y sus relaciones para descubrir errores.

    Existe un número de modelados para pruebas de comportamiento que pueden hacer uso de los grafos:

    - Modelado del flujo de transacción: Los nodos representan los pasos de alguna transacción (por ejemplo, los pasos necesarios para una reserva en una línea aérea usando un servicio

en línea), y los enlaces representan las conexiones lógicas entre los pasos (por ejemplo, vuelo.información.entrada es seguida de validación / disponibilidad.procesamiento).

- Modelado de estado finito: Los nodos representan diferentes estados del software observables por el usuario (por ejemplo, cada una de las pantallas que aparecen cuando un telefonista coge una petición por teléfono), y los enlaces representan las transiciones que ocurren para moverse de estado a estado (por ejemplo, petición-información se verifica durante inventario-disponibilidad-búsqueda y es seguido por cliente-factura-información-entrada).
- Modelado de flujo de datos: Los nodos objetos de datos y los enlaces son las transformaciones que ocurren para convertir un objeto de datos en otro.
- Modelado de planificación: Los nodos son objetos de programa y los enlaces son las conexiones secuenciales entre esos objetos. Los pesos de enlace se usan para especificar los tiempos de ejecución requeridos al ejecutarse el programa.
- Gráfica Causa-efecto: La gráfica Causa-efecto representa una ayuda gráfica en seleccionar, de una manera sistemática, un gran conjunto de casos de prueba. Tiene un efecto secundario beneficioso en precisar estados incompletos y ambigüedades en la especificación.

Un gráfico de causa-efecto es un lenguaje formal al cual se traduce una especificación. El gráfico es realmente un circuito de lógica digital (una red combinatoria de lógica), pero en vez de la notación estándar de la electrónica, se utiliza una notación algo más simple. No hay necesitad de tener conocimiento de electrónica con excepción de una comprensión de la lógica booleana (entendiendo los operadores de la lógica y, o, y no).

- **Partición equivalente:** Es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada. Típicamente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

El objetivo de partición equivalente es reducir el posible conjunto de casos de prueba en uno más pequeño, un conjunto manejable que evalúe bien el software. Se toma un riesgo porque se escoge no probar todo. Así que se necesita tener mucho cuidado al escoger las clases.

La partición equivalente es subjetiva. Dos probadores, quienes prueban un programa complejo pueden llegar a diferentes conjuntos de particiones. En el diseño de casos de prueba para partición equivalente se procede en dos pasos:

- Se identifican las clases de equivalencia: Las clases de equivalencia son identificadas tomando cada condición de entrada (generalmente una oración o una frase en la especificación) y repartiéndola en dos o más grupos. Es de notar que dos tipos de clases de equivalencia están identificados: las clases de equivalencia válidas representan entradas válidas al programa, y las clases de equivalencia inválidas representan el resto de los estados posibles de la condición (es decir, valores erróneos de la entrada).
- Se define los casos de prueba: El segundo paso es el uso de las clases de equivalencia para identificar los casos de prueba. El proceso es como sigue: se asigna un número único a cada clase de equivalencia. Hasta que todas las clases de equivalencia válidas han sido cubiertas por los casos de prueba, se escribe un nuevo caso de prueba que cubra la clase de equivalencia válida. Y por último hasta que los casos de prueba hayan cubierto todas las clases de equivalencia inválidas, se escribe un caso de la prueba que cubra una, y solamente una, de las clases de equivalencia inválidas descubiertas.

- **Análisis de valores límites:** Los errores tienden a darse más en los límites del campo de entrada que en el centro. Por ello, se ha desarrollado el análisis de valores límites (AVL) como técnica de prueba, la cual lleva a una elección de casos de prueba que ejerciten los valores límites. El análisis de valores límites es una técnica de diseño de casos de prueba que completa a la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva

a la elección de casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida.

- Condiciones sub-límite: Las condiciones límites normales son las más obvias de descubrir. Estas son definidas en la especificación o son evidentes al momento de utilizar el software. Algunos límites, sin embargo, son internos al software, no son necesariamente aparentes al usuario final pero aún así deben ser probadas por el probador. Estas son conocidas como condiciones sub-límites o condiciones límite internas.

- **Prueba de la tabla ortogonal:** Hay aplicaciones donde el número de parámetros de entrada es pequeño y los valores de cada uno de los parámetros están claramente delimitados. Cuando estos números son muy pequeños (por ejemplo, 3 parámetros de entrada tomando 3 valores diferentes), es posible considerar cada permutación de entrada y comprobar exhaustivamente el proceso del dominio de entrada. En cualquier caso, cuando el número de valores de entrada crece y el número de valores diferentes para cada elemento de dato se incrementa, la prueba exhaustiva se hace impracticable.

  La prueba de la tabla ortogonal puede aplicarse a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para posibilitar pruebas exhaustivas. El método de prueba de la tabla ortogonal es particularmente útil al encontrar errores asociados con fallos localizados (una categoría de error asociada con defectos de la lógica dentro de un componente software). La prueba de tabla ortogonal permite proporcionar una buena cobertura de pruebas con bastantes menos casos de prueba que en la estrategia exhaustiva.

- **Adivinando el error:** Dado un programa particular, se conjetura, por la intuición y la experiencia, ciertos tipos probables de errores y entonces se escriben casos de prueba para exponer esos errores. Es difícil dar un procedimiento para esta técnica puesto que es en gran parte un proceso intuitivo y ad hoc.

  La idea básica es enumerar una lista de errores posibles o de situaciones propensas a error y después escribir los casos de prueba basados en la lista. Por ejemplo, la presencia del valor 0 en la entrada de un programa es una situación con tendencia a error. Por lo tanto, puede ser que se

escriba los casos de prueba para los cuales los valores particulares de la entrada tienen valor 0 y para qué valores particulares de la salida se colocan de manera forzada a 0.

1.4.5 Tipos de Prueba.

- **Pruebas de Unidad:** Las pruebas de unidad tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente una vez que ha sido codificado. Es un proceso para probar los subprogramas, las subrutinas, los procedimientos individuales o las clases en un programa. Es decir, es mejor probar primero los bloques desarrollados más pequeños del programa, que inicialmente probar el software en su totalidad. Las motivaciones para hacer esto son tres:
    - Las pruebas de unidad son una manera de manejar los elementos de prueba combinados, puesto que se centra la atención inicialmente en unidades más pequeñas del programa.
    - La prueba de una unidad facilita la tarea de eliminar errores (el proceso de establecer claramente y de corregir un error descubierto), puesto que, cuando se encuentra un error, se sabe que existe en un módulo particular.
    - Las pruebas de unidad introducen paralelismo en el proceso de pruebas del software presentándose la oportunidad de probar los múltiples módulos simultáneamente.

  Se necesitan dos tipos de información al diseñar los casos de prueba para una prueba de unidad:
    - La especificación para el módulo
    - El código fuente del módulo.

  La especificación define típicamente los parámetros de entrada y de salida del módulo y su función.

  Las pruebas de unidad son en gran parte orientadas a caja blanca. Una razón es que como en pruebas de entidades más grandes tales como programas enteros (es el caso para los procesos de prueba subsecuentes), la prueba de caja blanca llega a ser menos factible. Una segunda razón es que los procesos de prueba subsecuentes están orientados a encontrar diversos tipos de errores. Por lo tanto, el procedimiento para el diseño de casos de prueba para una prueba de unidad es la siguiente: analizar la lógica del módulo usando uno o más de los métodos de caja

blanca y después completar los casos de prueba aplicando métodos de caja negra a la especificación del módulo.

- **Pruebas de Integración:** El objetivo de las pruebas de integración es verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes. Los tipos fundamentales de integración son los siguientes:
    - *Integración incremental:* Se combina el siguiente componente que se debe probar con el conjunto de componentes que ya están probados y se va incrementando progresivamente el número de componentes a probar.
    - *Integración no incremental:* Se prueba cada componente por separado y posteriormente se integran todos de una vez realizando las pruebas pertinentes.

    - *Pruebas Top-Down*: El primer componente que se desarrolla y prueba es el primero de la jerarquía (A). Los componentes de nivel más bajo se sustituyen por componentes auxiliares para simular a los componentes invocados. Una de las ventajas de aplicar esta estrategia es que las interfaces entre los distintos componentes se prueban en una fase temprana y con frecuencia.
    - Pruebas Bottom-up: En este caso se crean primero los componentes de más bajo nivel (E, F) y se crean componentes conductores para simular a los componentes que los llaman. A continuación se desarrollan los componentes de más alto nivel (B, C, D) y se prueban. Por último dichos componentes se combinan con el que los llama (A). Este tipo de enfoque permite un desarrollo más en paralelo que el enfoque de arriba - abajo, pero presenta mayores dificultades a la hora de planificar y de gestionar.

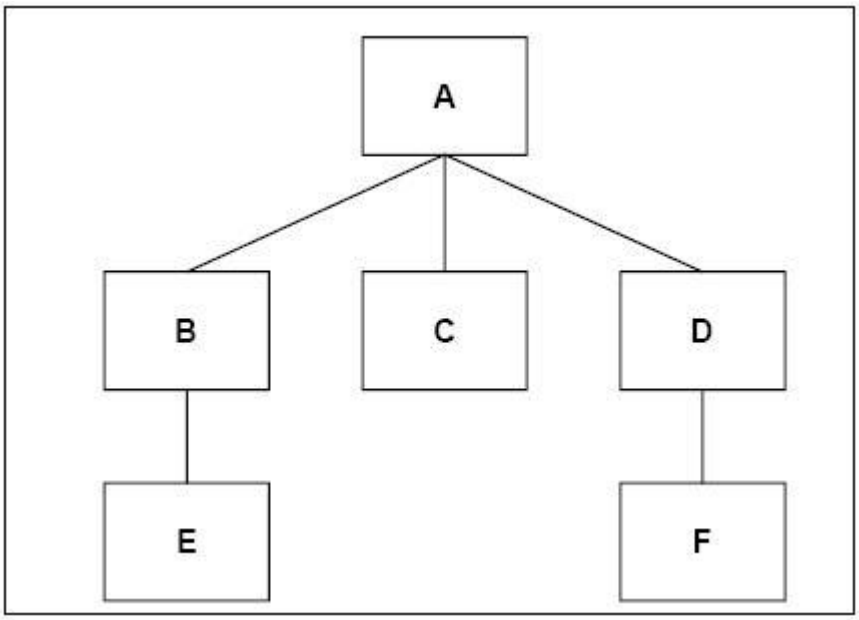

Fig. 1.4. Pruebas de integración.

- Estrategias combinadas: A menudo es útil aplicar las estrategias anteriores conjuntamente. De este modo, se desarrollan partes del sistema con un enfoque "top-down", mientras que los componentes más críticos en el nivel más bajo se desarrolla siguiendo un enfoque "bottom-up". En este caso es necesaria una planificación cuidadosa y coordinada de modo que los componentes individuales se encuentren en el centro.

- **Pruebas de Humo:** La prueba de humo es un método de prueba de integración que es comúnmente utilizada cuando se ha desarrollado un producto de software «empaquetado ». Es diseñado como un mecanismo para productos críticos por tiempo, permitiendo que el equipo de software valore su proyecto sobre una base sólida. Algunas de las actividades de la prueba de humo son las siguientes:

  - Los componentes que han sido traducidos a código se integran en una construcción.
  - Se diseña una serie de pruebas para descubrir errores que impiden a la construcción realizar su función adecuada.

- En la prueba de humo es habitual que la construcción se integre con otras construcciones y que se aplique una nueva prueba de humo al producto completo. La integración puede hacerse bien de forma descendente o ascendente.

- **Pruebas Funcionales:** La prueba funcional es un proceso para procurar encontrar discrepancias entre el programa y la especificación funcional. Una especificación funcional es una descripción exacta del comportamiento del programa desde el punto de vista del usuario final. La prueba funcional normalmente es una actividad de caja negra. Para realizar una prueba funcional, la especificación se analiza para derivar un sistema de casos de prueba utilizando las diferentes técnicas de caja negra. Se confía en el proceso de pruebas unitarias para alcanzar los criterios deseados de cobertura de caja blanca.

- **Pruebas de Sistema:** Las pruebas de sistema buscan discrepancias entre el programa y sus objetivos o requerimientos, enfocándose en los errores hechos durante la transición del proceso al diseñar la especificación funcional. Esto hace a las pruebas de sistema un proceso vital de pruebas, ya que en términos del producto, número de errores hechos, y severidad de esos errores, es un paso en el ciclo de desarrollo generalmente propenso a la mayoría de los errores.

  Las pruebas de sistema no son procesos para probar las funciones del sistema o del programa completo, porque esta sería redundante con el proceso de las pruebas funcionales. Las pruebas del sistema tienen un propósito particular: para comparar el sistema o el programa con sus objetivos originales (Requerimientos funcionales y no funcionales). Dado este propósito, se presentan dos implicaciones:

  - Las pruebas de sistema no se limitan a los sistemas. Si el producto es un programa, la prueba del sistema es el proceso de procurar demostrar cómo el programa, en su totalidad, no resuelve sus objetivos o requerimientos.
  - Las pruebas de sistema, por definición, son imposibles si no están los requerimientos por escrito, mensurables para el producto.

Las pruebas de sistema tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.

Son pruebas de integración del sistema de información completo, y permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Dan una visión muy similar a su comportamiento en el entorno de producción. Se distinguen los siguientes tipos de pruebas:

- *Pruebas de comunicaciones:* Determinan que las interfaces entre los componentes del sistema funcionan adecuadamente, tanto a través de dispositivos remotos, como locales. Asimismo, se han de probar las interfaces hombre/máquina.
- *Pruebas de rendimiento:* Consisten en determinar que los tiempos de respuesta están dentro de los intervalos establecidos en las especificaciones del sistema.
- *Pruebas de recuperación*: La prueba de recuperación es una prueba del sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente.
- *Pruebas de volumen:* Consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas.
- *Pruebas de sobrecarga:* Consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndole a cargas masivas. El objetivo es establecer los puntos extremos en los cuales el sistema empieza a operar por debajo de los requisitos establecidos.
- *Pruebas de tensión:* La prueba de tensión es poner al programa a grandes cargas o tensiones. Esto no se debe confundir con la prueba de volumen; una gran tensión es volumen máximo de datos, o de actividad, en un tiempo corto. Una analogía sería evaluar a un mecanógrafo. Una prueba de volumen se determinaría si el mecanógrafo hiciera frente a un bosquejo de un informe grande; una prueba de tensión se determinaría si el mecanógrafo puede mecanografiar a un índice de 50 palabras por minuto.

- *Pruebas de disponibilidad de datos:* Consisten en demostrar que el sistema puede recuperarse ante fallos, tanto de equipo físico como lógico, sin comprometer la integridad de los datos.
- *Pruebas de facilidad de uso:* Consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios, tanto para asegurar que se acomoda a su modo habitual de trabajo, como para determinar las facilidades que aporta al introducir datos en el sistema y obtener los resultados.
- *Pruebas de operación:* Consisten en comprobar la correcta implementación de los procedimientos de operación, incluyendo la planificación y control de trabajos, arranque y re-arranque del sistema, etc.
- *Pruebas de entorno:* Consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.
- *Pruebas de seguridad:* Consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos.
- *Pruebas de usabilidad:* Otra categoría importante de casos de prueba de sistema es la tentativa de encontrar problemas de factores humanos, o usabilidad. Sin embargo, un análisis de factores humanos sigue siendo una cuestión altamente subjetiva.
- *Pruebas de almacenamiento:* Los programas tienen de vez en cuando objetivos de almacenamiento que indiquen, por ejemplo, la cantidad de memoria principal y secundaria que el programa usa y el tamaño de los archivos temporales. Se diseñan casos de prueba para demostrar que estos objetivos de almacenaje no han sido encontrados.
- *Pruebas de configuración:* Programas tales como sistemas operativos, sistemas de gerencia de base de datos, y programas de conmutación de mensajes soportan una variedad de configuraciones de hardware, incluyendo varios tipos y números de dispositivos de entrada-salida y líneas de comunicaciones, o diversos tamaños de memoria. A menudo el número de configuraciones posibles es demasiado grande para probar cada uno, pero en lo posible, se debe probar el programa con cada tipo de dispositivo de hardware y con la configuración mínima y máxima. Si el programa por sí mismo se puede configurar para omitir componentes, o si puede funcionar en diversas computadoras, cada configuración posible de este debe ser probada.

- *Pruebas de instalación:* Algunos tipos de sistemas de software tienen complicados procedimientos de instalación. Las pruebas de los procedimientos de instalación es una parte importante del proceso de prueba del sistema. Esto es particular de un sistema automatizado de instalación que sea parte del paquete del programa. Al funcionar incorrectamente el programa de instalación podría evitar que el usuario tenga una experiencia acertada con el sistema. La primera experiencia de un usuario es cuando él o ella instalan la aplicación. Si esta fase se realiza mal, entonces el usuario/el cliente puede buscar otro producto o tener poca confianza en la validez de la aplicación.
- *Pruebas de la documentación:* Las pruebas de sistema también se refieren a la exactitud de la documentación del usuario. Una manera de lograr esto es utilizar la documentación para determinar la representación de los casos anteriores de prueba del sistema. Esto es, una vez se desea idear el caso de sobrecarga, se utilizaría la documentación como guía para escribir el caso de prueba real. También, la documentación del usuario debe ser el tema de una inspección, comprobándola para saber si hay exactitud y claridad. Cualquiera de los ejemplos ilustrados en la documentación se debe probar y hacer parte de los casos y alimentarlos al programa.

- **Pruebas de Implantación:** El objetivo de las pruebas de implantación es comprobar el funcionamiento correcto del sistema integrando el hardware y software en el entorno de operación, y permitir al usuario que, desde el punto de vista de operación, realice la aceptación del sistema una vez instalado en su entorno real y en base al cumplimiento de los requisitos no funcionales especificados.

- **Pruebas de Aceptación:** El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento. Las pruebas de aceptación son definidas por el usuario del sistema y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al usuario.

  La validación del sistema se consigue mediante la realización de pruebas de caja negra que demuestran la conformidad con los requisitos y que se recogen en el plan de pruebas, el cual

define las verificaciones a realizar y los casos de prueba asociados. Dicho plan está diseñado para asegurar que se satisfacen todos los requisitos funcionales especificados por el usuario teniendo en cuenta también los requisitos no funcionales relacionados con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema.

- *Pruebas alfa y beta*: Cuando se construye software a medida para un cliente, se lleva a cabo una serie de pruebas de aceptación para permitir que el cliente valide todos los requisitos. La mayoría de los desarrolladores de productos de software llevan a cabo un proceso denominado pruebas alfa y beta para descubrir errores que parezca que sólo el usuario final puede descubrir.

  - *Prueba alfa:* se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.
  - *Prueba beta:* se llevan a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.

- **Pruebas de Regresión:** El objetivo de las pruebas de regresión es eliminar el efecto onda, es decir, comprobar que los cambios sobre un componente de un sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.

  Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error como para realizar una mejora. No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también

es necesario controlar que las modificaciones no produzcan efectos negativos sobre el mismo u otros componentes. Las pruebas de regresión pueden incluir:

- La repetición de los casos de pruebas que se han realizado anteriormente y están directamente relacionados con la parte del sistema modificada.
- La revisión de los procedimientos manuales preparados antes del cambio, para asegurar que permanecen correctamente.
- La obtención impresa del diccionario de datos de forma que se compruebe que los elementos de datos que han sufrido algún cambio son correctos.

1.5 Realidad Actual de los Modelos de Prueba.

En la actualidad existen múltiples empresas de desarrollo de software centradas en la implantación de SOA. Para lograr este objetivo es necesario todo un proceso de calidad de inicio a fin, que incluye a su vez modelos y estrategias de pruebas de calidad. No existe un estándar entre estas empresas para el desempeño de dichos modelos y estrategias, cada una por su parte, ha creado el suyo propio y este a su vez, no es libremente compartido ya que son empresas propietarias. Todo esto es producto a que SOA es un tema novedoso a nivel internacional y la competencia entre las grandes compañías por ganar terreno en este campo se hace cada día mayor. Por los motivos antes mencionados y por la necesidad de tener en nuestro poder nuestro propio modelo de pruebas de calidad para una SOA, surge esta investigación y a su vez propuesta de modelo para implantar en nuestros proyectos.

1.6 Diferencias entre una arquitectura SOA y una arquitectura tradicional.

El enfoque SOA implica un cambio significativo respecto al modelo tradicional de TI ya que, en lugar de estructurar las aplicaciones a base de funciones, componentes y objetos, pasa a estructurarlas alrededor del concepto de servicios, unos servicios que pueden ser expuestos tanto interna como externamente mediante un conjunto de tecnologías estándar para su uso por otras aplicaciones propias y de terceros.
Una de las diferencias fundamentales entre SOA y OOP (Object Oriented Programming/Programación Orientada a Objetos) es la manera en la que ambas definen una “aplicación”. OOP determina que una aplicación está compuesta de clases interdependientes, mientras que SOA considera que una aplicación está compuesta por un conjunto de servicios autónomos. SOA, a diferencia de las arquitecturas de objetos

distribuidos como J2EE, refleja más fielmente los procesos y relaciones del mundo real. Es decir que SOA representa una manera más simple y natural de modelar y construir software que soluciona problemáticas de negocios del mundo real. Por ejemplo, los sistemas de objetos distribuidos están fuertemente acoplados y el modificar una parte de un sistema de este tipo implica romper alguna otra parte, salvo que esa otra parte sea también modificada al mismo tiempo. Este esquema ha comprobado ser poco práctico, especialmente cuando diferentes partes del sistema son desarrolladas y operadas por diferentes organizaciones como escenarios de B2B (Business To Business). Además la mayoría de estos sistemas de objetos distribuidos requieren que las diferentes partes del sistema sean creadas utilizando el mismo lenguaje de programación o arquitectura de objetos.

Los sistemas orientados a servicios en cambio, son diseñados con un bajo nivel de acoplamiento que facilita la implementación de cambios y estos servicios pueden ser desarrollados en cualquier lenguaje corriendo en diferentes plataformas. Desde el punto de vista de un usuario, la diferencia entre utilizar servicios y utilizar componentes tradicionales es imperceptible. Desde un punto de vista arquitectónico en cambio, podemos decir que los servicios a diferencia de los componentes, son autónomos, encapsulan sus propios datos y no forman parte, estrictamente hablando, de la aplicación, sino que son utilizados por ella. Estos servicios pueden ser generados por distintos equipos de desarrollo, en diferentes plataformas y en diferente espacio y tiempo, es decir que una aplicación puede ir creciendo y aumentando su funcionalidad a medida que se construyan nuevos servicios que no necesariamente deben estar bajo el control del equipo de desarrollo de la aplicación, por lo que es esencial que entre nuestras aplicaciones y los servicios que ellas consuman haya un mínimo acoplamiento.

Los servicios pueden ser externos o internos, podemos utilizar servicios disponibles en Internet para, por ejemplo, validar tarjetas de crédito o mostrar en una GUI (Graphical User Interface/Interfaz de Usuario Gráfica) de nuestra aplicación la cotización del dólar o permitirle a nuestros usuarios utilizar algún motor de búsqueda como Google, pero también podemos hacer que nuestra aplicación sea un conjunto de servicios, por ejemplo el servicio de crear un cliente, de facturar, de obtener el saldo de una cuenta, de esa manera no solo estamos adhiriéndonos al concepto de SOA, sino que estamos dejando abierta una puerta para que futuras aplicaciones de nuestra compañía o externas, hagan uso de esa funcionalidad con un mínimo de esfuerzo. Nuestros servicios pueden a su vez hacer uso de servicios pre-existentes.

1.7 Conclusiones.

En este capítulo se sustentó teóricamente el trabajo realizado y se establecieron las bases para el desarrollo del modelo de pruebas, al explicarse los conceptos relacionados con dicho modelo y realizándose un análisis detallado de los mismos. Se abordaron temas de gran importancia para el futuro entendimiento del modelo de prueba desarrollado para una arquitectura SOA, para ello fue de vital valor conocer la realidad actual de dichos modelos en el mundo para poder llegar a conclusiones con respecto al contenido tratado y las diferencias que existen entre una arquitectura SOA y cualquier otra arquitectura de desarrollo.

CAPÍTULO 2: SOLUCIÓN PROPUESTA.

2.1 Introducción.

En este capítulo se realiza la solución propuesta del Modelo de Prueba, compuesto por la metodología a usar durante el desarrollo de las pruebas del sistema completo, así como los entregables que recogerán todos los datos de las fases del mismo. También se describen los diagramas que se proponen para estructurar los elementos de dicho modelo y se muestran además los roles que participan en la realización de las pruebas para una Arquitectura Orientada a Servicios (SOA). Al final del capítulo se muestran indicadores para la selección de herramientas de calidad.

2.2 Descripción del Modelo V.

El Modelo V que fuera desarrollado de manera simultánea, pero independiente, en Alemania (por el Ministerio de Defensa) y los Estados Unidos (por el Consejo Nacional de Ingeniería de Sistemas) a finales de 1980, es una evolución del modelo de desarrollo en Cascada. El modelo en cascada consiste en la ejecución secuencial de una serie de fases que se suceden, lo que da nombre al modelo. Cada fase genera documentación para la siguiente, esta documentación debe ser aprobada. Una fase no comienza hasta que la anterior ha terminado, además requiere disponer de unos requisitos completos y precisos al principio del desarrollo. (20)

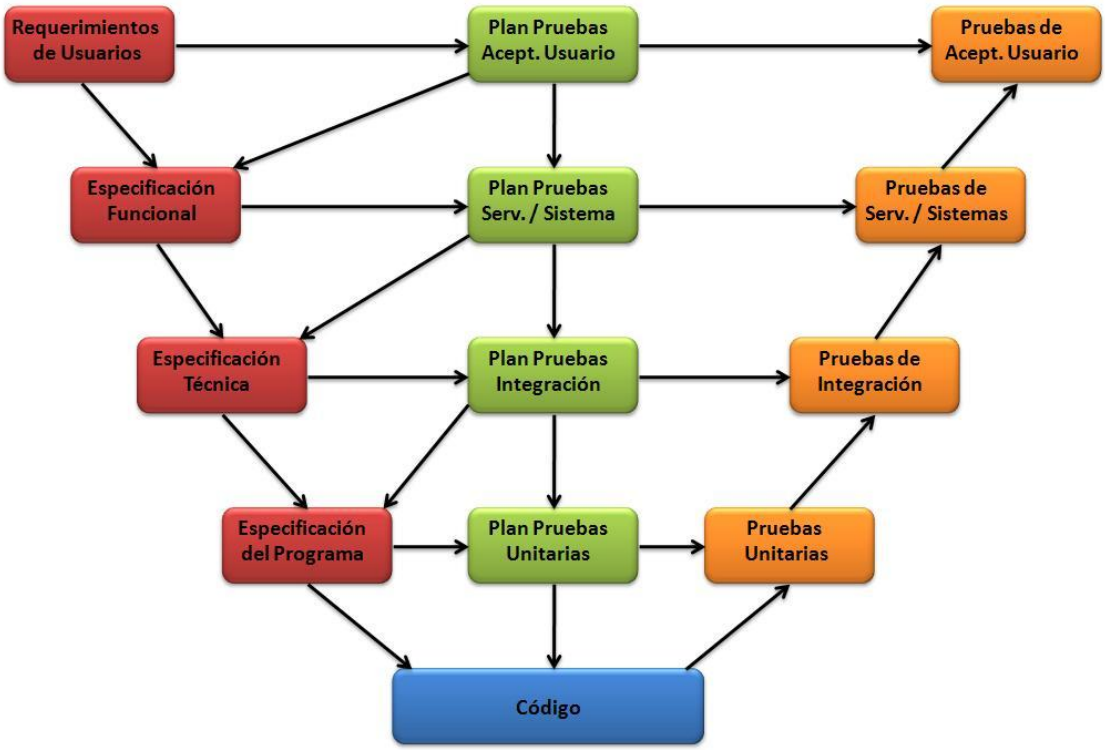

Fig. 2.1. Modelo V.

La primera mitad del Modelo V es similar al Modelo en Cascada y la otra mitad tiene como finalidad hacer pruebas e integración asociado a cada una de las etapas de la mitad anterior. Esta es una de las ventajas fundamentales que presenta con respecto a otros modelos, pues involucra chequeos de cada una de las etapas, considerando las pruebas como un proceso que corre en paralelo con el análisis y el desarrollo, a lo largo de todo el ciclo de vida del proyecto, en lugar de constituir una fase aislada al final del proyecto.

También tiene la facilidad que en el momento en el cual se está realizando una fase, es posible realizar también la documentación y planeación para las pruebas que se realizarán más adelante. En la representación gráfica clásica del Modelo V, las fases de desarrollo de software aparecen a la izquierda y los correspondientes niveles de pruebas a la derecha. Partiendo de los requisitos o del diseño (a la izquierda) se deben planificar y preparar los niveles de pruebas correspondientes (a la derecha).

En general, cada actividad de pruebas a la derecha valida la actividad enfrentada de la izquierda. Además, los niveles superiores de pruebas en el diagrama están basados en caja negra (pruebas basadas en las especificaciones) y los niveles inferiores están basados en caja blanca (basadas en la estructura interna de los componentes del sistema), teniendo dichos niveles por separado, distintos objetivos, entornos y perfiles de personal.

El Modelo V es simplemente animar al equipo de proyecto a determinar continuamente cómo probar satisfactoriamente los resultados del proyecto y cada organización puede utilizar su versión, basándolo en su propia terminología. Definitivamente se trata de un modelo más robusto y completo que el Modelo en Cascada, y puede producir software de mayor calidad. El Modelo V es una adecuada metodología de prueba para entregar proyectos SOA por las siguientes razones:

- Estimula una metodología descendente con respecto a la definición de los requisitos del proceso de negocio, de alto nivel del diseño técnico funcional, de seguridad, etc.
- Estimula un método de prueba ascendente:

    - Prueba funciones individuales dentro de un servicio.
    - Prueba un servicio individual.
    - Prueba un conjunto compuesto de servicios a través de pruebas de un proceso integrado.
    - Prueba un sistema de negocio completo.

- En SOA los servicios son ligeramente acoplados y es por eso que un modelo de prueba ascendente es recomendable.
- Los niveles reflejan diferentes puntos de vista de las pruebas en los diferentes niveles de detalles.
- El Modelo V estimula las pruebas a lo largo de todo el ciclo de vida de desarrollo del software.

2.3 Estrategia de pruebas en SOA.

Propósito

Probar una arquitectura SOA podría verse como un complejo problema de la informática. Para cualquier problema complejo, lo principal es dividirlo en pequeños problemas, en componentes más manejables y

fomentar la calidad en estos. Las bases para unas pruebas satisfactorias en una arquitectura SOA son las siguientes:

- Igualdad de distribución del esfuerzo de las pruebas en todo el ciclo de vida del proyecto. Muchas organizaciones aún no reconocen los beneficios reales de las técnicas de revisión estáticas y formales durante las primeras fases del proyecto. La mayoría o la totalidad del esfuerzo en las pruebas llega demasiado tarde al final del ciclo de vida del proyecto. Se precisará de más esfuerzo en las pruebas del nivel de servicio.
- El equipo de pruebas SOA es una combinación de expertos del dominio de negocio y de la tecnología.
- Diseñar la estrategia de pruebas del proyecto junto a los requerimientos técnicos y de negocio del mismo.
- Presupuesto para el equipo de pruebas para que participen desde el inicio del proyecto.
- Implementar controles de calidad en todo el ciclo de vida del proyecto.
- Las pruebas de seguridad no son una actividad de fin del proyecto. Diseñar y planificar las pruebas de seguridad desde el inicio del proyecto.
- Las herramientas de prueba son una necesidad.

2.3.1 ¿Cómo se prueba una arquitectura SOA?

¿Cómo se prueba una arquitectura SOA? Pues no existe una definición estándar para hacerlo. En cambio podemos aprender a descomponer la arquitectura en sus componentes, trabajando desde los más simples hasta los más complejos, probando cada uno de ellos. En otras palabras, se tiene que dividir la arquitectura en dominios como los servicios, seguridad y gobierno y probar cada dominio por separado usando la estrategia y herramientas definidas por los indicadores propuestos en el epígrafe 2.8.

SOA está formada por servicios de aplicación débilmente acoplados y altamente interoperables y un enfoque de pruebas en SOA debe seguir el mismo patrón.

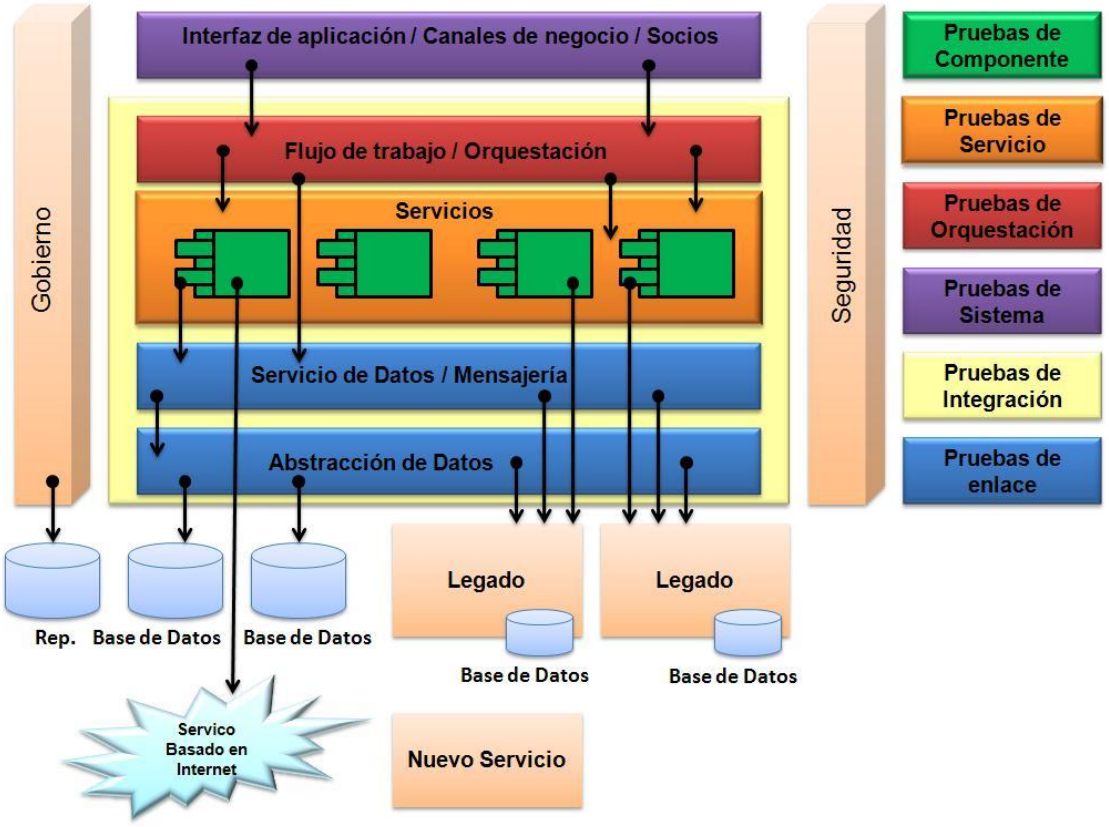

Fig. 2.2. Componentes de pruebas aplicados a SOA.

La figura anterior representa un modelo de los componentes de pruebas aplicados a SOA y como estos están interrelacionados. El equipo de pruebas diseña la estrategia de pruebas del proyecto y los planes deben tener un macro entendimiento de cómo todos los componentes trabajan tanto independiente como colectivamente.

2.4 Fases o niveles de pruebas y tipos de pruebas.

Se definen las pruebas en SOA en las siguientes fases o niveles:

- Nivel de prueba de Componentes de Servicios.
- Nivel de prueba de *Servicios*.
- Nivel de prueba de *Integración*.

- Nivel de prueba de *Proceso/Orquestación*.
- Nivel de prueba de *Sistema.*

Nivel de prueba de Componentes de Servicios.

En el nivel de pruebas de Componentes de Servicios, las pruebas de Unidad son normalmente realizadas por los desarrolladores para probar que el código no solo compile satisfactoriamente, sino que la funcionalidad básica de los componentes y funciones dentro de un servicio estén trabajando como lo especificado. El objetivo primario de las pruebas de Componentes de Servicios es tomar pequeños pedazos de software probable en la aplicación, aislarlo del código restante, y determinar si el comportamiento es justamente como se esperaba. Cada componente es probado por separado antes de integrarlo dentro de un servicio o servicios. Las siguientes actividades de calidad y prueba son recomendadas en esta fase/nivel de prueba:

- Revisiones formales de código para asegurarse que cumple con los estándares de la organización y para identificar cualquier función potencial y defectos de seguridad o debilidad.
- Los criterios de calidad de entrada y salida no son solo definidos para este nivel de prueba, sino que son alcanzados antes de moverse al siguiente nivel de pruebas.

Nivel de prueba de Servicios.

Las pruebas de Servicios constituyen la fase/nivel más importante dentro de la Metodología de Pruebas en una SOA. Las siguientes actividades de calidad y prueba son recomendadas en esta fase/nivel de prueba:

- Revisiones formales de código para asegurarse que cumple con los estándares de la organización y para identificar cualquier función potencial y defectos de seguridad o debilidad.
- Pruebas funcionales, de rendimiento y seguridad son ejecutadas contra los servicios. Esto exigirá la ayuda de herramientas de pruebas automatizadas.
- Los criterios de calidad de entrada y salida no son solo definidos para este nivel de prueba, sino que son alcanzados antes de moverse al siguiente nivel de pruebas.

Las pruebas del nivel de servicio deben garantizar que el servicio no solo cumpla los requisitos del proyecto actual, sino que es más importante todavía el cumplimiento de los requerimientos operacionales y de negocio de los otros procesos que están usando ese servicio.

Nivel de prueba de Integración.

La fase de pruebas de Integración se enfocará en las interfaces de los servicios. Esta fase de prueba apunta a determinar si el comportamiento de la interfaz y la información compartida entre los servicios, está trabajando como lo especificado. El equipo de prueba asegurará que todos los servicios entregados a esta fase de pruebas cumplen con lo definido en la definición de la interfaz, en términos de normas, la validación del formato y los datos. Los escenarios de prueba de las pruebas de Integración también deberían trabajar las capas de comunicaciones y los protocolos de red.

Nivel de prueba de Proceso/Orquestación.

Las pruebas de Proceso/Orquestación aseguran que los servicios estén funcionando colectivamente como lo especificado. Esta fase de pruebas cubriría la lógica del negocio, la ordenación en secuencia, manipulación de excepciones y descomposición de procesos (incluyendo rehúso de servicios y procesos).

Nivel de prueba de Sistema.

Las pruebas del Nivel de Sistema formarán la mayoría, si no todas las Fases de Prueba de Aceptación de Usuario. Esta fase probará que la solución técnica de la arquitectura SOA ha dado los requerimientos del negocio definido y ha encontrado los criterios de aceptación del negocio definido.

La siguiente tabla muestra las fases o niveles de pruebas y los tipos de pruebas que se realizan en cada fase o nivel.

Fases de prueba	Documentos guía	Funcionales	Rendimiento	Interoperabilidad	Compatibilidad con Versiones Anteriores	Aceptación	Seguridad
Nivel de Componente	Esquema Técnico, Especificaciones y Normas de Aplicación	✓	✓	✓	✓	✓	✓
Nivel de Servicio	Requerimientos del Negocio, Esquema Técnico & Políticas y Normas de Gobierno	✓	✓	-	-	✓	✓
Integración & Orquestación	Requerimientos del Negocio, Esquema Técnico & Políticas y Normas de Gobierno	✓	-	✓	✓	-	✓
Sistema/Proceso (Aceptación de usuario)	Requerimientos del Negocio, Esquema Técnico & Políticas y Normas de Gobierno	✓	✓	✓	✓	✓	✓

Pruebas de Gobierno.

El Gobierno SOA es un factor fundamental en el éxito de cualquier implementación SOA. Es también el término más “débilmente” usado, ya que abarca todo el ciclo de vida de la implementación SOA, desde el diseño hasta el tiempo de ejecución en curso de mantenimiento. El Gobierno SOA se refiere a los

Estándares y Políticas que gobiernen el diseño, construcción y puesta en práctica de una solución SOA y las políticas que deben aplicarse durante el tiempo de ejecución.

Las organizaciones deben tener bien definido el diseño, el desarrollo, las pruebas y las normas de seguridad que guiarán y dirigirán implementaciones SOA. Los controles de calidad y revisiones deben llevarse a cabo a través de todo el ciclo de vida del proyecto y los procesos, para garantizar el cumplimiento. La contraparte adecuada debe conducir esas revisiones y los cambios con respecto a los parámetros estándar de las normas recomendadas que deben ser acordadas por el equipo de gobierno de la organización. Los siguientes son ejemplos de tipos de políticas del gobierno SOA:

- Políticas de calidad de los servicios en rendimiento, seguridad y transacciones.
- Políticas de regulación.
- Políticas de negocio (reglas).
- Políticas de auditoría (qué eventos necesitan ser registrados, cuánto tiempo se debe mantener un evento).
- Políticas de infraestructura (acceso, copias de respaldo, recuperación de desastres y rescate de la conexión).

Los casos de prueba serán construidos y ejecutados en todas las fases de prueba del proyecto para determinar si las políticas SOA se están aplicando. Las políticas SOA pueden ser aplicadas en tiempo de ejecución, mediante el uso de tecnologías y/o herramientas de monitoreo. Las pruebas del gobierno SOA no estarán en una fase de prueba separada. Las pruebas que el gobierno SOA aplicará tomarán lugar en todo el ciclo de vida del proyecto, a través de revisiones homólogas y diferentes escenarios de prueba que serán ejecutados durante las fases de pruebas separadas.

Pruebas de Seguridad.

Como SOA evoluciona y crece dentro de su organización, el perfil y la necesidad de pruebas de seguridad aumentarán. Hoy, varias organizaciones realizan una inadecuada cantidad de pruebas de penetración al final de un proyecto. SOA en combinación con el gobierno y el cumplimiento normativo, requerirá actividades de pruebas de seguridad que serán incorporadas dentro del ciclo de vida completo del proyecto.

Muchos procesos de negocio dentro de una organización se componen de varios servicios, físicamente ubicados en diferentes partes de la red corporativa, actualizando una serie de bases de datos y datos potencialmente sensibles compartidos con organizaciones externas. Esto hace que nos planteemos la pregunta de "¿Qué tan segura es la información ya que navega en una compleja red tanto interna como externa?".

Hoy muchas organizaciones realizan pruebas de penetración de la seguridad al final del ciclo de vida del proyecto para cubrir toda la seguridad del software. Las pruebas de penetración son un ensayo autorizado de violación de la seguridad de un sistema usando un agente externo y/o técnico de acceso gusano. Realizando pruebas de penetración en el final de un proyecto corre un riesgo significativo no solo de encontrar graves errores de seguridad sino también de la entrega de un sistema que tiene un inadecuado diseño de seguridad.

Como SOA evoluciona, las redes de las organizaciones llegarán a ser más complejas y no será posible proteger a todos los activos. La prioridad de seguridad será requerida para proteger los activos más valiosos de la compañía.

- La definición de requerimientos del negocio debe incluir requerimientos de seguridad.
- Una evaluación de riesgos de seguridad debe realizarse durante la fase de diseño técnico para priorizar y justificar las pruebas de seguridad requeridas.
- Revisar formalmente todos los resultados técnicos que se han construidos de acuerdo a las normas de seguridad definidas por sus organizaciones.
- Las pruebas de penetración de seguridad pueden ser planificadas y ejecutadas en el *nivel de componentes de servicios* y no solo cuando ha sido entregado un sistema totalmente integrado.

Hoy, hay disponibles muchas herramientas de pruebas de seguridades tanto comerciales como libres. Estas herramientas han evolucionado de dispositivos de exploración que informan de los posibles puntos débiles de seguridad a las herramientas que realmente ejecutan específicos tipos de pruebas de penetración. Las herramientas de seguridad son necesarias si su organización desea un acreditado y repetible método de pruebas de seguridad.

Pruebas Funcionales.

Las pruebas funcionales o pruebas de caja negra determinarán si un componente o servicio o la totalidad del sistema está operando según las especificaciones sin hacer referencia a la técnica de funcionamiento interno o el diseño. Los requerimientos de negocio y las definiciones de un alto nivel de diseño técnico son los principales insumos para el diseño de casos de pruebas funcionales.

Rendimiento.

Como SOA crece y evoluciona con el tiempo, y muchos de los componentes y servicios de SOA son rehusados, será fundamental que de cada uno de estos componentes y servicios se conozca su rendimiento y capacidad de producción bajo carga, así como la forma en que son escalables. Las organizaciones deben abandonar el mal pensamiento “solo puedes hacer pruebas de rendimiento, de carga y de estrés en una solución técnica completamente integrada”, para probar el rendimiento a servicios individuales y a componentes de la arquitectura SOA. Muchas de las nuevas herramientas de prueba para SOA soportan pruebas de rendimiento de componentes y servicios. Los servicios deben ser entregados a la fase de pruebas de integración con el rendimiento y las declaraciones de capacidad probados y definidos.

Interoperabilidad.

La interoperabilidad es la capacidad de un sistema o un producto para trabajar con otros sistemas o productos sin un esfuerzo especial por parte del cliente. Los servicios lograrán interoperabilidad ya sea por estar estrictamente adherido a estándares de interfaz publicados o por el uso de un agente de servicios que convertirá los datos al formato de otra interfaz de servicios rápidamente.

Las pruebas de interoperabilidad en tiempo de diseño no son suficientes. Las pruebas de interoperabilidad en tiempo de ejecución son también necesarias para SOA. Las pruebas en tiempo de diseño global combinadas con el comportamiento activo de las pruebas de interoperabilidad en tiempo de ejecución, asegura que los activos de TI se pueden integrar independiente de la plataforma, el sistema operativo y el lenguaje de programación.

Compatibilidad con versiones anteriores.

Las pruebas de compatibilidad con versiones anteriores determinarán si los cambios en la interfaz afectarán a los usuarios existentes (también conocidos como consumidores de servicios) de la interfaz. Si los usuarios existentes no se ven afectados entonces el cambio es compatible con versiones anteriores. Si existen usuarios que se ven afectados entonces el cambio no es compatible con versiones anteriores, y una solución o estrategia será necesaria para gestionar el impacto del cambio. Una interfaz de servicio en algún momento tendrá que cambiar. Cualquier cambio hecho a una interfaz debe ser evaluado y probado para su compatibilidad con versiones anteriores.

Aceptación.

El gobierno será un factor fundamental en la implementación satisfactoria de una SOA. El gobierno en SOA está comprometido por las normas de la organización y por las políticas. Las pruebas de aceptación deben llevarse a cabo durante todo el ciclo de vida del proyecto para garantizar que estas normas y políticas se apliquen. El gobierno y el cumplimiento de las normas también demandan este tipo de pruebas.

Seguridad.

SOA requiere pruebas de seguridad para ser diseñada y planificada correctamente desde el inicio del proyecto. Las pruebas de seguridad deben ser ejecutadas a lo largo de las fases de prueba del proyecto y no solo cuando el sistema completo ha sido entregado al final del ciclo de vida.

Estrategia de regresión.

Las pruebas de regresión también conocidas como pruebas de validación proporcionan una validación repetible, coherente, de cada cambio a los servicios en desarrollo o que están siendo modificados. Cada vez que un defecto es reparado, existe la posibilidad de introducir nuevos errores, problemas y defectos inadvertidamente. Un elemento de incertidumbre se presenta sobre la capacidad del servicio para repetir todo lo que ha ido bien hasta el punto del fallo. Las pruebas de regresión son las pruebas selectivas de un servicio o sistema SOA que ha sido modificado, para garantizar previamente que no fallen los servicios de trabajo como resultado de las reparaciones.

Las pruebas de regresión no prueban que determinado defecto ha sido reparado. El objetivo de las pruebas de regresión es garantizar que el servicio o sistema, hasta el punto de reparación, no ha sido afectado negativamente por el arreglo o revisión. Las organizaciones deben invertir en el desarrollo y mantenimiento de las suites funcionales y no funcionales (rendimiento y seguridad). Se recomienda que una parte importante de esas suites deban estar destinadas a los servicios claves y no solo a la totalidad de sistemas integrados. Esto será esencial si es cierto que se ha de llevar a cabo el rehúso de los servicios. Las herramientas de pruebas automatizadas serán una de las dependencias de la rentabilidad de la ejecución y el mantenimiento de dichas suites de regresión.

Pruebas de aceptación de usuario.

Las pruebas de aceptación de usuario son un hecho cuando se implementan nuevos sistemas o procesos. Es la manera formal por la cual el nuevo sistema o proceso realmente cumple los asuntos fundamentales y los requerimientos operacionales. Hoy, muchas organizaciones, por una variedad de razones, solo involucran los principales stakeholders del negocio al comienzo del proyecto para definir los requerimientos de negocio y al final del proyecto para llevar a cabo las pruebas de aceptación antes de la implementación.

Esto aumenta el riesgo de que los usuarios requieran que se repitan y dupliquen muchas pruebas durante esta fase y se detecten defectos de alta gravedad y requerimientos perdidos muy tarde en el ciclo de vida del proyecto. Los principales interesados del negocio y los usuarios necesitarán estar activamente más involucrados a lo largo de todo el ciclo de vida del proyecto. Necesitan saber cómo se construye la calidad y las pruebas dentro de todo proyecto y no solo al final. Necesitarán involucrarse activamente y entender completamente en detalle el aumento del esfuerzo circundante de las pruebas dando un servicio individual dentro de la fase de Pruebas de Aceptación de Usuarios.

La fase de Pruebas de Aceptación de Usuarios debe ser corta y dirigida a determinar si la solución se adapta con el propósito y no dejar probar todo nuevamente. Un principio fundamental de SOA es que el negocio dirigirá la SOA, no la tecnología. El principal desafío para el equipo de pruebas será construir un puente hacia los principales usuarios del negocio y stakeholders operacionales para garantizar su participación activa a lo largo de todo el proyecto.

Pruebas basadas en riesgo.

Como SOA crece y evoluciona dentro de su organización, con muchos procesos de negocio rehusando muchos servicios, será posible definir un número infinito de escenarios de prueba para cada proyecto SOA. Este es el principal desafío que debe ser vencido si su organización se propone implementar SOA exitosamente. Muchas organizaciones ahora están adoptando métodos de pruebas basadas en riesgo. ¿Qué es una prueba basada en riesgo? Una definición simple podría ser que los scripts y casos de prueba planificados para ser ejecutados, son justificados y priorizados por las implicaciones financieras comprendidas y acordadas para el negocio en fallo y la probabilidad de ocurrencia de algún fallo.

2.5 Modelo de Pruebas Integrado.

Este modelo ofrece soluciones reales a los nuevos desafíos de las pruebas en SOA, precisando significantes esfuerzos y actividades de prueba en el nivel de servicio. La principal razón detrás de dicha afirmación es que los servicios serán reutilizados. Si un servicio tiene defectos conocidos así como otras cuestiones de calidad, entonces probablemente no será seleccionado para ser reutilizado por el equipo de desarrollo. SOA exigirá que los servicios individuales sean entregados a las fases de pruebas de integración y aceptación de usuario con las declaraciones y las garantías de calidad en la funcionalidad, rendimiento y seguridad del negocio.

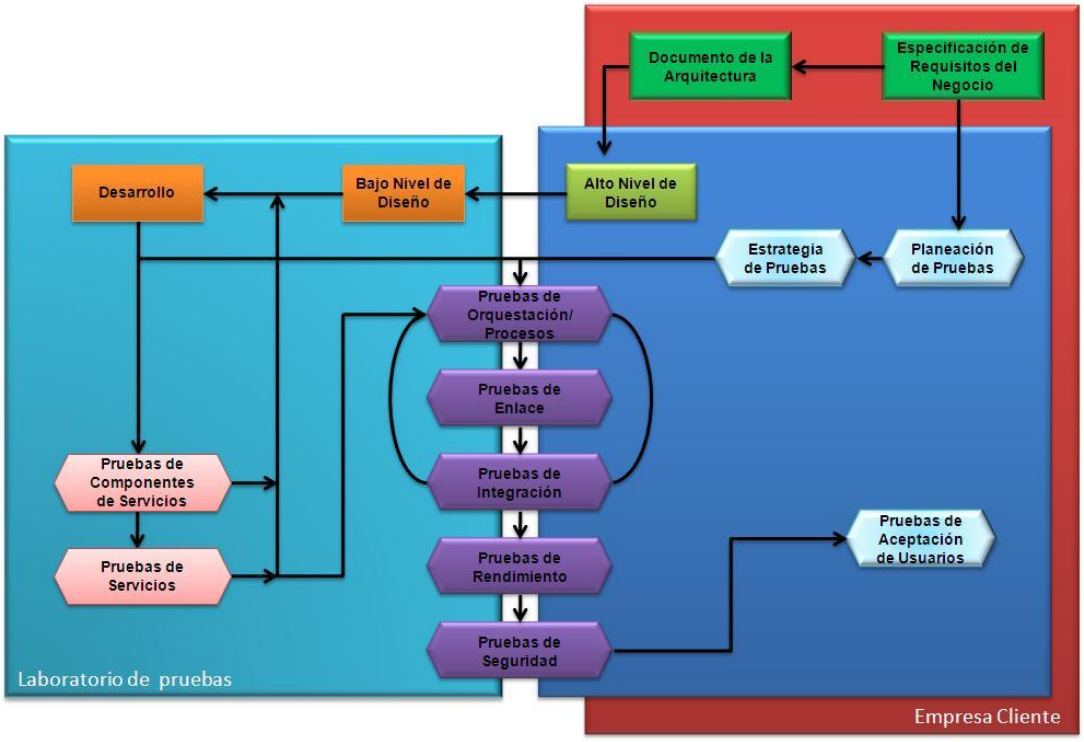

Fig. 2.3. Modelo de Pruebas Integrado.

La figura anterior representa una vista general del proceso de desarrollo de las pruebas de calidad en una arquitectura orientada a servicios. El modelo se encuentra dividido en dos partes, una parte que tendría lugar en la empresa cliente y la otra en el laboratorio de pruebas. Existen actividades como la especificación de los requisitos, la planeación y estrategia de pruebas, así como las pruebas de aceptación de usuarios (últimas a realizar durante el proceso de desarrollo) que son obligatorias realizarlas en el área del cliente. En cambio, parte del diseño, el mayor peso del desarrollo del sistema, las pruebas de componentes de servicios y las pruebas de servicios, son actividades realizadas en el laboratorio de pruebas. Existen también un conjunto de pruebas que pueden ser llevadas a cabo en ambos lugares, como son el caso de las pruebas de orquestación, de enlace, de integración, de rendimiento y de seguridad.

2.6 Roles.

Debido a que deben realizarse pruebas a cada nivel de la arquitectura, no existe algún rol que esté a cargo de todas las pruebas. En cambio, hay múltiples roles, al menos a cargo o responsable de las pruebas en su nivel. Estos roles deben coordinar y comunicarse con los probadores de los niveles a su alrededor. Definir el equipo de pruebas es un paso crítico a la hora de realizar pruebas en SOA. Hay técnicas de pruebas que son únicas a las pruebas en SOA que requieren de un conjunto específico de habilidades en el equipo de pruebas. Los roles que no deben faltar en un equipo de pruebas en SOA son:

- **Jefe de Proyecto de Pruebas**: Es el responsable del éxito de la prueba, este rol es el defensor de las pruebas y de la calidad, planifica y administra los recursos, resuelve los problemas que impidan las pruebas.
- **Probador**: Ejecuta casos o escenarios de pruebas y documenta la ejecución. Es el responsable de las pruebas de unidad, integración y sistema, lo que incluye ejecutar las pruebas, evaluar su ejecución, recuperar los errores y garantizar los resultados de las pruebas.
- **Analista de pruebas**: Es el responsable de identificar y definir las pruebas requeridas, monitorear el progreso de las mismas y el resultado en cada ciclo de pruebas, evaluando la calidad total experimentada como un resultado de las actividades de prueba.
- **Diseñador de Pruebas**: Es el responsable de definir el método de prueba y asegurar su implementación exitosa. El rol incluye identificar técnicas apropiadas, herramientas e instrucciones para implementar las pruebas necesarias y encauzar los recursos correspondientes para las pruebas.

Otros roles que son de importancia para el desarrollo exitoso de las pruebas y su continuación son:

- **Consultor de pruebas**: Puede realizar labor de consultoría en diversas áreas, como Metodología, Automatización, Infraestructura o Soporte de Negocio.
- **Especialista en automatización de pruebas**: Tiene un conocimiento alto de los aspectos “teóricos” de las pruebas para las que se usan sus herramientas. Tiene conocimientos de desarrollo y/o arquitectura. Maneja una o varias herramientas de pruebas.
- **Gestor de Calidad**: Se encarga del control de la calidad de los entregables y de la definición y obtención de métricas de proceso y de producto.

Cada uno de estos roles tiene unas actividades asignadas dentro de un proyecto. Para desempeñar esas tareas requiere de conocimientos y de experiencia.

2.7 Documentos entregables.

Para poder verificar que los requisitos funcionales levantados al inicio del proyecto sean válidos y cumplan con las expectativas del cliente se hace necesario realizar pruebas a lo largo del ciclo de vida del software, pero no solo basta con esta importante tarea, también es necesario dejar escrito cada una de las entradas y salidas realizadas, los defectos encontrados a lo largo de las pruebas, los responsables que intervinieron y así una serie de indicadores que demuestren la realidad del asunto. A continuación se muestran los entregables que no deben faltar en la realización de las pruebas.

2.7.1 Plan de Pruebas.

Es la colección formada por los casos de prueba y procedimientos de prueba. Este entregable incluye el propósito de las pruebas, qué elemento se va a probar, las herramientas a utilizar y con qué recursos, así como el documento que va a ser entregado. Al tener el resultado de las pruebas se puede comparar lo obtenido con lo esperado. En este artefacto también se reflejan las características de hardware y software que serán empleados para realizar el conjunto de las pruebas al sistema.

2.7.2 Casos de Pruebas.

Este entregable define un conjunto de datos de entradas, condiciones de ejecución y resultados esperados de las pruebas, identificados para hacer una evaluación de los aspectos específicos de un elemento objeto de prueba. Cada Caso de Prueba está asociado a un escenario de un Caso de Uso en particular. Los casos de prueba deben ser escritos con el detalle suficiente para que el probador pueda empezar rápidamente a ejecutar pruebas y a encontrar defectos. Además, estos reflejan trazabilidad con los casos de uso, las especificaciones suplementarias de requerimientos y diseño del sistema, garantizando que los procedimientos de pruebas sean compatibles con las necesidades de los usuarios/clientes.

En la metodología los Casos de Uso dirigen todo el proceso de desarrollo, es por ello que los Casos de Uso se transforman en un activo que puede directamente conducir el proceso de pruebas. Un Caso de

Prueba no es igual a un Caso de Uso. El Caso de Prueba extiende o amplía la información contenida en un Caso de Uso.

2.7.3 Documento de las No conformidades.

Este entregable registra los defectos y dificultades detectados durante el proceso de pruebas.

2.8 Indicadores para la selección de herramientas de pruebas.

Para el desarrollo de una SOA es necesaria e imprescindible la realización de pruebas a través de herramientas que faciliten este proceso, ya que hay que probar de principio a fin todo el ciclo de desarrollo y sería muy conveniente automatizar parte de este trabajo. Existe un conjunto de herramientas que se usan en el entorno de las pruebas sobre una Arquitectura Orientada a Servicios, tanto de código abierto como propietarias, pero no todas son 100% efectivas ni todas hacen todo lo que necesitamos. A continuación aparece un conjunto de indicadores que ayudarán a definir qué herramientas usar y cuáles no para el trabajo dentro de un equipo de pruebas. De manera general, cubren gran parte de las funcionalidades que se requieren en una buena herramienta de prueba para un ambiente SOA.

- **Alerta de las interfaces de prueba**: estandarizar las interfaces y los mensajes para las pruebas.
- **Automatización de pruebas basadas en mensajes**: grabar, reproducir y gestionar los scripts de prueba.
- **Virtualización**: la capacidad de simulación virtual de los proveedores y consumidores de servicios.
- **Simulación**: la capacidad de simular las aplicaciones como parte de una prueba de regresión.
- **Prueba de carga**: la capacidad para aplicaciones de pruebas de estrés.
- **Validación**: verificación de fallos en los niveles de componentes (envío de mensajes) y de aplicación (almacenamiento de datos).
- **Componentes**: visibilidad de componentes Java y/o .NET.
- **Introspección**: soporte WSDL y XML para generar los datos y operaciones de las pruebas.
- **Administración**: manejar los casos de prueba, scripts, datos y resultados de las mismas.
- **Seguridad**: SSL (Secure Sockets Layer, protocolo criptográfico que provee seguridad e integridad en los datos para comunicaciones sobre redes TCP/IP como la internet), WS-Security (Seguridad

en Servicios WEB, protocolo de comunicaciones que suministra un medio para aplicar seguridad a los Servicios Web), así como firmas digitales.

- **Formato de datos específicos**: EDI (Electronic Data Interchange, se refiere a la estructura de transmisión de datos entre organizaciones por medios electrónicos), HL7 (Health Level Seven, es un conjunto de estándares para el intercambio electrónico de información médica).
- **Prueba continua**: construir, desplegar y probar de forma automatizada.

2.9 Conclusiones.

En este capítulo se ha obtenido un modelo de prueba para su utilización en proyectos desarrollados en una Arquitectura Orientada a Servicios. El mismo está compuesto principalmente por la metodología a usar durante el desarrollo de las pruebas, los entregables que recogerán todos los datos de cada fase y los roles involucrados. También se definieron indicadores para la selección de herramientas de pruebas. El resultado final puede ser usado en cualquier proyecto productivo que siga una línea SOA/BPM.

CAPÍTULO 3: VALIDACIÓN.

3.1 Introducción.

A veces se hace necesario tener el criterio de otras personas con el conocimiento necesario para saber si lo investigado es realmente cercano a la realidad. En este capítulo se pretende realizar la validación del modelo de pruebas para una arquitectura SOA, utilizando el método Delphi, donde se trabaja con la ayuda de un grupo de Expertos.

3.2 Método Delphi.

El primer estudio de Delphi fue realizado en 1950 por la Rand Corporation para la fuerza aérea de Estados Unidos, y se le dio el nombre de Proyecto Delphi. El método Delphi utiliza como fuente de información un grupo de personas a las que se supone un conocimiento elevado de la materia que se va a tratar. La calidad de los resultados depende:

- De la elaboración de los cuestionarios.
- La elección de los expertos consultados.

El Delphi es uno de los métodos de pronosticación más confiables, constituye un procedimiento para confeccionar un cuadro de la evolución de situaciones complejas, a través de la elaboración estadística de las opiniones de un grupo de expertos en el tema tratado. Permite rebasar el marco de las condiciones actuales más señaladas de un fenómeno y alcanzar una imagen integral y más amplia de su posible evolución, reflejando las valoraciones individuales de los expertos que pueden estar basadas en un análisis lógico, como en su experiencia intuitiva. El método presenta 4 características principales:

Anonimato: Ningún experto conoce la identidad de los otros que componen el grupo de debate. Esto tiene una serie de aspectos positivos, como son:

- Impide la posibilidad de que un miembro del grupo sea influenciado por la reputación de otro de los miembros o por el peso que supone oponerse a la mayoría. La única influencia posible es la de la congruencia de los argumentos.

- Permite que un miembro pueda cambiar sus opiniones sin que eso suponga una pérdida de imagen.
- El experto puede defender sus argumentos con la tranquilidad que da saber que en caso de que sean erróneos, su equivocación no va a ser conocida por los otros expertos.

Iteración y realimentación controlada: La iteración se consigue al presentar varias veces el mismo cuestionario. Como además, se van presentando los resultados obtenidos con los cuestionarios anteriores, se consigue que los expertos vayan conociendo los distintos puntos de vista y puedan ir modificando su opinión si los argumentos presentados les parecen más apropiados que los suyos.

Respuesta del grupo en forma estadística: La información que se presenta a los expertos no es sólo el punto de vista de la mayoría, sino que se presentan todas las opiniones indicando el grado de acuerdo que se ha obtenido.

Heterogeneidad: Pueden participar expertos de determinadas ramas de actividad sobre las mismas bases.

Algunas de las ventajas que presenta el método Delphi son:

- Permite obtener información de puntos de vista sobre temas muy amplios o muy específicos. Los ejercicios Delphi son considerados “holísticos”, cubriendo una variedad muy amplia de campos.
- El horizonte de análisis puede ser variado.
- Permite la participación de un gran número de personas, sin que se forme el caos.
- Ayuda a explorar de forma sistemática y objetiva problemas que requieren la concurrencia y opinión cualificada.
- Elimina o aminora los efectos negativos de las reuniones de grupo “Cara-Cara”.

Debido a lo anterior es que se ha decidido el uso del método Delphi. Para aplicar el método se siguieron tres etapas fundamentales.

3.2.1 Elección de Expertos.

Se definen una serie de criterios de selección de los expertos, como por ejemplo:

- Graduado del Nivel Superior.
- Vinculación al desarrollo de proyectos productivos.
- Conocimientos sobre pruebas de calidad del software.
- Conocimientos sobre Arquitectura Orientada a Servicios (SOA).

Para la selección de los expertos finales se hace necesario primeramente conocer el grado de conocimiento del experto en cuestión, la misma se realiza con la ayuda del Coeficiente de competencia. Este coeficiente se determina mediante la fórmula: K= ½ (kc + ka), donde kc es el coeficiente de conocimientos y ka es el coeficiente de argumentación. El coeficiente de conocimientos se obtiene de la siguiente tabla que recoge una autoevaluación del posible experto.

Tabla No.1

								X		
0	1	2	3	4	5	6	7	8	9	10

El presunto experto marcará en la casilla enumerada, según su criterio acerca de la capacidad que él tiene sobre el tema que se la ha sometido a su consideración, en una escala del 0 al 10 y que después para ajustarla a la teoría de las probabilidades se multiplicará por 0,1; de esta forma, la evaluación "0" indica que el experto no tiene absolutamente ningún conocimiento de la problemática correspondiente, mientras que la evaluación "10" significa que el experto tiene pleno conocimiento de la problemática tratada. Entre estas dos evaluaciones extremas hay nueve intermedias. En la tabla considerada, kc = 0.8.

Para calcular el *coeficiente de argumentación* se procede de la siguiente forma:

Tabla No.2

FUENTES DE ARGUMENTACION	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M (medio)	B (bajo)
Análisis realizados por usted.			
Experiencia.			
Trabajos de autores nacionales.			
Trabajos de autores extranjeros.			
Su propio conocimiento del tema.			
Su intuición.			

En esta tabla el experto debe marcar, según su criterio, su grado de competencia sobre los aspectos sometidos a consideración. Las marcas de los expertos se traducen a puntos, según la siguiente escala:

Tabla No.3

FUENTES DE ARGUMENTACION	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M (medio)	B (bajo)
Análisis realizados por usted.	0.3	0.2	0.1
Experiencia.	0.5	0.4	0.2
Trabajos de autores nacionales.	0.05	0.05	0.05
Trabajos de autores extranjeros.	0.05	0.05	0.05
Su propio conocimiento del tema.	0.05	0.05	0.05
Su intuición.	0.05	0.05	0.05
Totales	1.0	0.8	0.5

Con estos elementos es suficiente para obtener el coeficiente de competencia K. Por ejemplo, si las selecciones del experto en la tabla son las siguientes:

Tabla No. 4

FUENTES DE ARGUMENTACION	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M (medio)	B (bajo)
Análisis realizado por usted.	x		
Experiencia.		x	
Trabajos de autores nacionales.	x		
Trabajos de autores extranjeros.		x	
Su propio conocimiento del tema.		x	
Su intuición.		x	

Entonces **ka** = 0,3 + 0,4 + 4(0.05) = 0,9

El código para la interpretación de tales coeficientes de competencia es el siguiente:

Si 0.8 < k <1.0, el coeficiente de competencia es alto.

Si 0.5 < k < 0.8, el coeficiente de competencia es medio.

Si k < 0.5 el coeficiente de competencia es bajo.

Observación: Como a la categoría de "bajo" se le otorgaron puntos, siempre el coeficiente de competencia quedará comprendido entre

$$\frac{0+0.5}{2} \leq K \leq \frac{1+1}{2} <==> 0.25 \leq K \leq 1$$

Como puede apreciarse el coeficiente de competencia del experto analizado es alto pues 0.8 < 0.9 < 1.0. La forma descrita con anterioridad nos permite seleccionar la competencia de nuestros expertos. Se lograron seleccionar de acuerdo a este análisis realizado un total de 11 expertos. En la siguiente figura se representa el resultado de acuerdo al coeficiente de competencia en general:

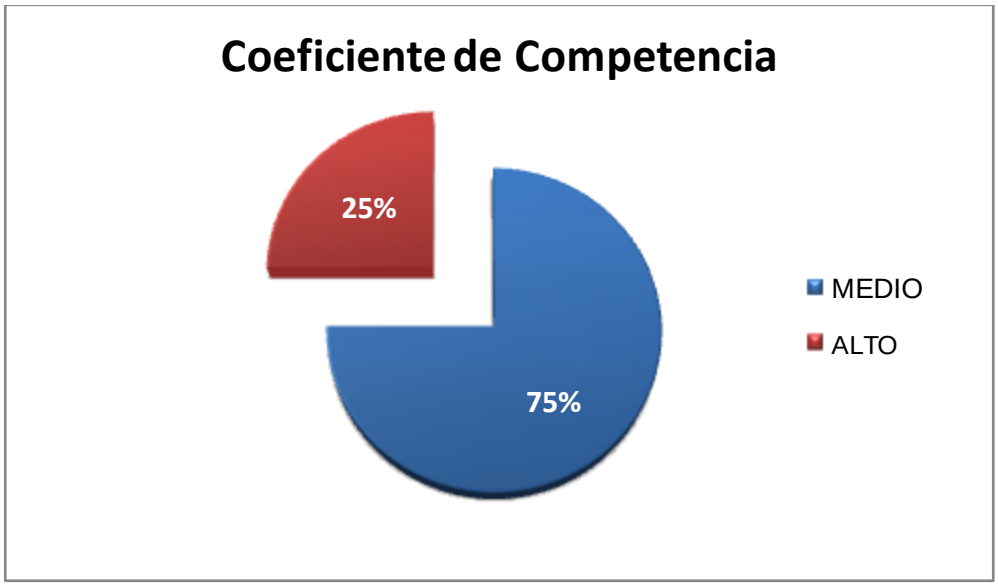

Fig. 3.1. Coeficiente de competencia.

Logrado ya el número de expertos, se buscan sus criterios sobre la temática sometida a consideración. Las preguntas a formular no deben ser demasiadas, pero si sobre cuestiones medulares sobre la investigación que realizamos.

3.2.2 Desarrollo práctico y explotación de resultados.

A continuación se ilustran las 35 afirmaciones Ai sobre el modelo de prueba para una arquitectura SOA realizada a 11 expertos. Se pidió que evaluaran los pasos en las categorías Ci de muy adecuada, bastante adecuada, adecuada, poco adecuada y no adecuada. Se confeccionan tablas para ir recogiendo los resultados aportados por los expertos. Para ello es necesario auxiliarse del programa Microsoft Excel 2003. Los resultados se recogen en una tabla de doble entrada como la siguiente:

Tabla No. 5

Tabla de Frecuencias Acumuladas:							
No	**Elementos**	**C1**	**C2**	**C3**	**C4**	**C5**	**Total**
1	A1	0	4	7	0	0	**11**
2	A2	5	5	1	0	0	**11**
3	A3	3	4	4	0	0	**11**
4	A4	3	6	2	0	0	**11**
5	A5	6	3	2	0	0	**11**
6	A6	8	2	1	0	0	**11**
7	A7	10	1	0	0	0	**11**
8	A8	8	3	0	0	0	**11**
9	A9	7	4	0	0	0	**11**
10	A10	6	3	2	0	0	**11**
11	A11	8	0	3	0	0	**11**
12	A12	10	1	0	0	0	**11**
13	A13	9	1	1	0	0	**11**
14	A14	5	3	3	0	0	**11**
15	A15	6	3	2	0	0	**11**
16	A16	8	3	0	0	0	**11**
17	A17	7	2	2	0	0	**11**
18	A18	11	0	0	0	0	**11**
19	A19	11	0	0	0	0	**11**
20	A20	9	1	0	1	0	**11**
21	A21	9	1	1	0	0	**11**
22	A22	8	1	2	0	0	**11**
23	A23	9	1	1	0	0	**11**
24	A24	5	3	3	0	0	**11**
25	A25	5	4	2	0	0	**11**
26	A26	5	3	3	0	0	**11**
27	A27	11	0	0	0	0	**11**
28	A28	11	0	0	0	0	**11**
29	A29	8	1	2	0	0	**11**
30	A30	3	4	4	0	0	**11**
31	A31	2	6	3	0	0	**11**
32	A32	5	5	1	0	0	**11**
33	A33	9	1	1	0	0	**11**
34	A34	5	2	4	0	0	**11**
35	A35	6	4	1	0	0	**11**
Total de aspectos a evaluar		**35**					

La tabla de frecuencias acumuladas quedaría representada gráficamente como lo demuestra la figura:

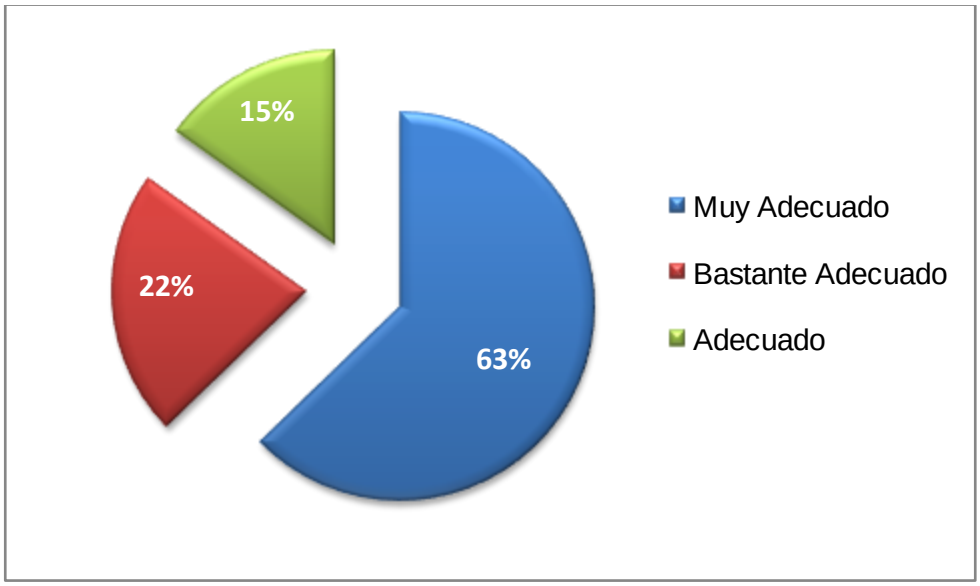

Fig. 3.2. Representación gráfica de la tabla de frecuencias acumuladas.

Tabulados los datos, se realizan los siguientes pasos para obtener los resultados deseados:

Primer paso: Se construye una tabla de frecuencias acumuladas. Esto es, cada número en la fila, excepto el primero se obtiene sumándole el anterior:

Tabla No. 6

Tabla de frecuencias absolutas acumuladas:						
No	**Elementos**	**C1**	**C2**	**C3**	**C4**	**C5**
1	A1	0	4	11	11	11
2	A2	5	10	11	11	11
3	A3	3	7	11	11	11
4	A4	3	9	11	11	11
5	A5	6	9	11	11	11
6	A6	8	10	11	11	11
7	A7	10	11	11	11	11
8	A8	8	11	11	11	11
9	A9	7	11	11	11	11

10	A10	6	9	11	11	11
11	A11	8	8	11	11	11
12	A12	10	11	11	11	11
13	A13	9	10	11	11	11
14	A14	5	8	11	11	11
15	A15	6	9	11	11	11
16	A16	8	11	11	11	11
17	A17	7	9	11	11	11
18	A18	11	11	11	11	11
19	A19	11	11	11	11	11
20	A20	9	10	10	11	11
21	A21	9	10	11	11	11
22	A22	8	9	11	11	11
23	A23	9	10	11	11	11
24	A24	5	8	11	11	11
25	A25	5	9	11	11	11
26	A26	5	8	11	11	11
27	A27	11	11	11	11	11
28	A28	11	11	11	11	11
29	A29	8	9	11	11	11
30	A30	3	7	11	11	11
31	A31	2	8	11	11	11
32	A32	5	10	11	11	11
33	A33	9	10	11	11	11
34	A34	5	7	11	11	11
35	A35	6	10	11	11	11

Observación: En la frecuencia acumulativa desaparece la última columna.

Segundo paso: Se copia la tabla anterior y se borran los resultados numéricos. Ahora, en esta nueva tabla, se construye la tabla de frecuencias relativas acumulativas. Esta tabla se logra dividiendo por 31 (número total de expertos) cada uno de los números de la tabla anterior. En esta tabla queda eliminada una columna pues hay 5 categorías y sólo se necesitan cuatro puntos de corte (con cuatro puntos se obtienen 5 intervalos).

Tabla No. 7

No	Elementos	C1	C2	C3	C4
1	A1	0.0001	0.3636	0.9999	0.9999
2	A2	0.45454545	0.9091	0.9999	0.9999
3	A3	0.27272727	0.6364	0.9999	0.9999
4	A4	0.27272727	0.8182	0.9999	0.9999
5	A5	0.54545455	0.8182	0.9999	0.9999
6	A6	0.72727273	0.9091	0.9999	0.9999
7	A7	0.90909091	0.9999	0.9999	0.9999
8	A8	0.72727273	0.9999	0.9999	0.9999

9	A9	0.63636364	0.9999	0.9999	0.9999
10	A10	0.54545455	0.8182	0.9999	0.9999
11	A11	0.72727273	0.7273	0.9999	0.9999
12	A12	0.90909091	0.9999	0.9999	0.9999
13	A13	0.81818182	0.9091	0.9999	0.9999
14	A14	0.45454545	0.7273	0.9999	0.9999
15	A15	0.54545455	0.8182	0.9999	0.9999
16	A16	0.72727273	0.9999	0.9999	0.9999
17	A17	0.63636364	0.8182	0.9999	0.9999
18	A18	0.9999	0.9999	0.9999	0.9999
19	A19	0.9999	0.9999	0.9999	0.9999
20	A20	0.81818182	0.9091	0.9091	0.9999
21	A21	0.81818182	0.9091	0.9999	0.9999
22	A22	0.72727273	0.8182	0.9999	0.9999
23	A23	0.81818182	0.9091	0.9999	0.9999
24	A24	0.45454545	0.7273	0.9999	0.9999
25	A25	0.45454545	0.8182	0.9999	0.9999
26	A26	0.45454545	0.7273	0.9999	0.9999
27	A27	0.9999	0.9999	0.9999	0.9999
28	A28	0.9999	0.9999	0.9999	0.9999
29	A29	0.72727273	0.8182	0.9999	0.9999
30	A30	0.27272727	0.6364	0.9999	0.9999
31	A31	0.18181818	0.7273	0.9999	0.9999
32	A32	0.45454545	0.9091	0.9999	0.9999
33	A33	0.81818182	0.9091	0.9999	0.9999
34	A34	0.45454545	0.6364	0.9999	0.9999
35	A35	0.54545455	0.9091	0.9999	0.9999

Tercer paso: Buscar las imágenes de los elementos de la tabla anterior por medio de la función (Dist. Normal. Standard Inv). La siguiente tabla se muestra con los resultados obtenidos de los pasos anteriores, donde se agregan tres nuevas columnas y una fila para colocar los valores de la suma de las columnas y de las filas (**Suma**); el promedio de las filas (**P**); el valor de **N** (se obtiene al dividir la suma de las sumas entre 175, este 175 se ha obtenido de multiplicar el número de categorías (5) por el número de preguntas); el valor **N-P** (da el valor promedio que otorgan los expertos consultados a cada pregunta de la metodología propuesta).

Tabla No. 8

							N=	**1.91**
No	**Elementos**	**C1**	**C2**	**C3**	**C4**	**Suma**	**P**	**N-P**
1	A1	-3.72	-0.35	3.72	3.72	3.37	**1.07**	**-3.72**
2	A2	-0.11	1.34	3.72	3.72	8.66	**-0.26**	**-0.11**
3	A3	-0.60	0.35	3.72	3.72	7.18	**0.11**	**-0.60**
4	A4	-0.60	0.91	3.72	3.72	7.74	**-0.03**	**-0.60**

5	A5	0.11	0.91	3.72	3.72	8.46	**-0.21**	**0.11**
6	A6	0.60	1.34	3.72	3.72	9.38	**-0.44**	**0.60**
7	A7	1.34	3.72	3.72	3.72	12.49	**-1.21**	**1.34**
8	A8	0.60	3.72	3.72	3.72	11.76	**-1.03**	**0.60**
9	A9	0.35	3.72	3.72	3.72	11.51	**-0.97**	**0.35**
10	A10	0.11	0.91	3.72	3.72	8.46	**-0.21**	**0.11**
11	A11	0.60	0.60	3.72	3.72	8.65	**-0.25**	**0.60**
12	A12	1.34	3.72	3.72	3.72	12.49	**-1.21**	**1.34**
13	A13	0.91	1.34	3.72	3.72	9.68	**-0.51**	**0.91**
14	A14	-0.11	0.60	3.72	3.72	7.93	**-0.07**	**-0.11**
15	A15	0.11	0.91	3.72	3.72	8.46	**-0.21**	**0.11**
16	A16	0.60	3.72	3.72	3.72	11.76	**-1.03**	**0.60**
17	A17	0.35	0.91	3.72	3.72	8.70	**-0.26**	**0.35**
18	A18	3.72	3.72	3.72	3.72	14.88	**-1.81**	**3.72**
19	A19	3.72	3.72	3.72	3.72	14.88	**-1.81**	**3.72**
20	A20	0.91	1.34	1.34	3.72	7.30	**0.08**	**0.91**
21	A21	0.91	1.34	3.72	3.72	9.68	**-0.51**	**0.91**
22	A22	0.60	0.91	3.72	3.72	8.95	**-0.33**	**0.60**
23	A23	0.91	1.34	3.72	3.72	9.68	**-0.51**	**0.91**
24	A24	-0.11	0.60	3.72	3.72	7.93	**-0.07**	**-0.11**
25	A25	-0.11	0.91	3.72	3.72	8.23	**-0.15**	**-0.11**
26	A26	-0.11	0.60	3.72	3.72	7.93	**-0.07**	**-0.11**
27	A27	3.72	3.72	3.72	3.72	14.88	**-1.81**	**3.72**
28	A28	3.72	3.72	3.72	3.72	14.88	**-1.81**	**3.72**
29	A29	0.60	0.91	3.72	3.72	8.95	**-0.33**	**0.60**
30	A30	-0.60	0.35	3.72	3.72	7.18	**0.11**	**-0.60**
31	A31	-0.91	0.60	3.72	3.72	7.13	**0.13**	**-0.91**
32	A32	-0.11	1.34	3.72	3.72	8.66	**-0.26**	**-0.11**
33	A33	0.91	1.34	3.72	3.72	9.68	**-0.51**	**0.91**
34	A34	-0.11	0.35	3.72	3.72	7.67	**-0.01**	**-0.11**
35	A35	0.11	1.34	3.72	3.72	8.89	**-0.31**	**0.11**
Suma		**19.63**	**56.48**	**127.78**	**130.17**	**334.05**		
Punto de corte		**0.56**	**1.61**	**3.65**	**3.72**			

Las sumas obtenidas en las cuatro primeras columnas nos dan los puntos de cortes: 0.56, 1.61, 3.65, 3.72. Los puntos de corte nos sirven para determinar la categoría o grado de adecuación de cada paso de la metodología según la opinión de los expertos consultados. La tabla que se muestra a continuación representa los puntos de cortes por las preguntas realizadas a los expertos, evidenciando en qué rango se encuentra la pregunta con respecto al punto de corte para determinar si la misma es muy adecuada, bastante adecuada, adecuada o poco adecuada.

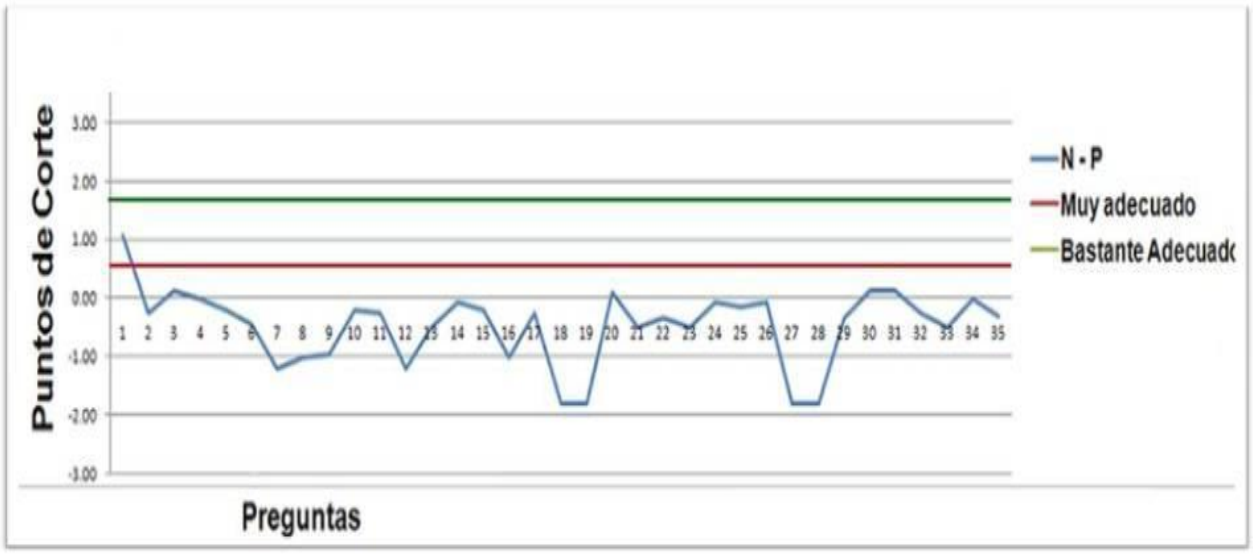

Fig. 3.3. Puntos de Corte.

Las preguntas que resultarían muy adecuado serían las que poseen valores menores que 0.56; bastante adecuado los valores entre (0.56, 1.61); adecuado entre (1.61, 3.65); poco adecuado (3.65, 3.72); y no adecuado más de 3.72. De acuerdo a los valores de N-P se tiene:

Menos de 0.56; (0.56, 1.61); (1.61, 3.65); (3.65, 3.72); más de 3.72.

Se obtiene finalmente que:

AFIRMACIONES	CATEGORÍAS
1	Bastante Adecuado
2	Muy Adecuado
3	Muy Adecuado
4	Muy Adecuado
5	Muy Adecuado
6	Muy Adecuado
7	Muy Adecuado
8	Muy Adecuado
9	Muy Adecuado
10	Muy Adecuado
11	Muy Adecuado

12	Muy Adecuado
13	Muy Adecuado
14	Muy Adecuado
15	Muy Adecuado
16	Muy Adecuado
17	Muy Adecuado
18	Muy Adecuado
19	Muy Adecuado
20	Muy Adecuado
21	Muy Adecuado
22	Muy Adecuado
23	Muy Adecuado
24	Muy Adecuado
25	Muy Adecuado
26	Muy Adecuado
27	Muy Adecuado
28	Muy Adecuado
29	Muy Adecuado
30	Muy Adecuado
31	Muy Adecuado
32	Muy Adecuado
33	Muy Adecuado
34	Muy Adecuado
35	Muy Adecuado

De acuerdo a los resultados obtenidos mostrados en la tabla anterior se puede dar por concluida la validación en cuanto a su elaboración teórica, ya que los resultados arrojados fueron satisfactorios. Se presenta a continuación un resumen con los resultados obtenidos.

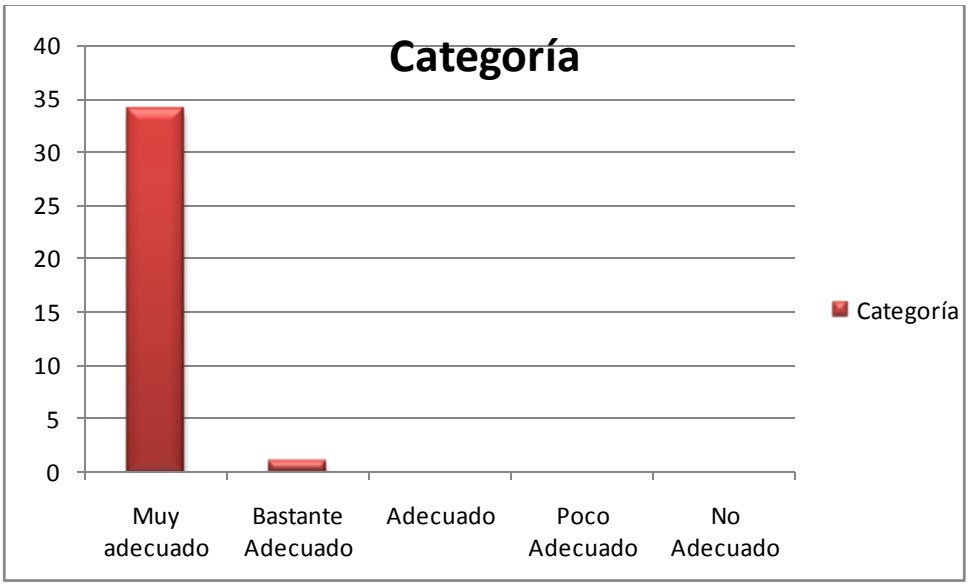

Fig. 3.4. Resultados generales de la encuesta.

3.3 Conclusiones.

En este capítulo se validó la solución propuesta realizada del modelo de prueba para una arquitectura SOA, utilizando el método de validación Delphi. Se seleccionaron 11 expertos y se validaron 35 afirmaciones relacionadas con el modelo planteado, los resultados obtenidos fueron satisfactorios evaluados de bastante adecuado y muy adecuado.

CONCLUSIONES.

El presente trabajo finaliza definiéndose un modelo flexible y acorde a los requisitos necesarios para la realización de pruebas de calidad y lo más importante es que es un modelo que puede ser utilizado por cualquier empresa o proyecto de desarrollo que siga una línea SOA/BPM. El mismo está compuesto por la metodología a usar durante el desarrollo de las pruebas, los entregables que recogerán todos los datos de cada fase, los roles involucrados y además se definieron indicadores para la selección de herramientas de pruebas. Para la validación de la solución propuesta se utilizó el método Delphi, con la ayuda de un grupo de expertos, donde los resultados obtenidos fueron satisfactorios.

RECOMENDACIONES.

Probar en la práctica el modelo de prueba propuesto en proyectos que utilicen una arquitectura orientada a servicios.

Utilizar el modelo de prueba propuesto en otras empresas que sigan una línea SOA/BPM.

Continuar el estudio del modelo de prueba para adecuarlo con la evolución de SOA.

REFERENCIAS BIBLIOGRÁFICAS.

[1] Gartner. [Online] http://gartner.com.

[2] Revista-ays. [Online] www.revista-ays.com/DocsNum08/SOA/roncero.pdf.

[3] **2008.** SOA agenda. [Online] Agosto 9, 2008. http://soaagenda.com/journal/articulos/category/soa/.

[4] [Online] https://developer.mozilla.org/es/Servicios_Web_XML.

[5] Thinkink. [Online] http://thinkink.es/index.php?option=com_content&view=article&id=158%3Ala-revolucion-del-xml&catid=61%3Aautomatizacion&Itemid=97&lang=es.

[6]Gxtechnical.[Online]
http://www.gxtechnical.com/gxdlsp/pub/GeneXus/Internet/TechnicalPapers/Web_Services.htm.

[7] Desarrollo Web. [Online] http://www.desarrolloweb.com/articulos/1857.php.

[8] **Kiran Garimella, Michael Lees, Bruce Williams.** *Introducción a BPM pra Dummies.* Indianápolis, Indiana : Wiley Publishing, Inc., 2008. 978-0-470-37359-0 : s.n.

[9] **Martinez, Marcela.** Sonic Software. [Online]

[10] Cicese. [Online] http://telematica.cicese.mx/internetII/qcudi/qos_cudi.html.

[11] **María Pérez, Luis E. Mendoza, Anna C. Grimán.** [Online]
http://www.willydev.net/InsiteCreation/v1.0/descargas/soa/willydev_estimaciondecalidadws.pdf.

[12] Contratosinformaticos. [Online] http://www.contratosinformaticos.com/sla/.

[13] **T., José Camilo Daccach.** Gestiopolis. [Online] http://www.gestiopolis.com/delta/term/TER433.html.
The Software Experts. [Online] http://www.the-software-experts.de/e_dta-sw-process.htm.

[14] [Online] http://www.crmagil.com/Catalogos/KnowledgeSync/FolletoKS2008.pdf.

[15] IEEE. [Online] http://www.ieee.org/portal/site.

[16] **Rubio, Gabriel Buades.** UIB. [Online] http://dmi.uib.es/bbuades/calidad/sld001.htm.

[17] **Lovelle, Juan Manuel Cueva.** [Online] http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF.

[18] Software Ingineering Institute. [Online] http://www.sei.cmu.edu/cmmi/general/index.html.

[19] **ROJAS, JOHANNA ROJAS and BARRIOS, EMILIO JOSÉ. 2007.** udistrital. [Online] 2007.
http://www.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/index.html.

[20] [Online] http://rguerrero334.blogspot.es/img/Modelos_de_procesos_del_software.pdf.

BIBLIOGRAFÍA.

1. **María José, Roca V. 2005.** *Pruebas de Integración de Productos: Un enfoque práctico*. Cali : ParqueSoft, 2005.
2. **Lisandra Rodríguez Ferrer, Frankis Peña Valera. 2008.** *Propuesta de métricas de calidad para el desarrollo de aplicaciones en la UCI con una Arquitectura Orientada a Servicios.* Universidad de las Ciencias Informáticas. 2008. Tesis.
3. **Palomo, Miguel Ángel García and Elcuera, Mamdouh.** Sistedes. [Online] http://www.sistedes.es/TJISBD/Vol-1/No-4/articles/pris-07-garcia-mceps.pdf.
4. The Software Experts. [Online] http://www.the-software-experts.de/e_dta-sw-process.htm.
5. [Online] http://maestros.its.mx/jcabrera/Calidad/Unidad4.pdf.
6. **Parmenter, David.** gestiopolis. [Online] http://www.gestiopolis.com/administracion-estrategia/revision-kpi-key-performance-indicators.htm.
7. [Online] http://aps.sld.cu/bvs/materiales/meto-investigacion/Cap%EDtulo%201.html.
8. **Quintana, Ing. Rolando. Diciembre de 2007**. *Propuesta de indicadores para medir competencias del personal según el rol en proyectos multimedia*. La Habana : s.n., Diciembre de 2007. Tesis.

GLOSARIO DE TÉRMINOS.

- **Ad Hoc:** Es una locución latina que significa literalmente «para esto». Generalmente se refiere a una solución elaborada específicamente para un problema o fin preciso y, por tanto, no es generalizable ni utilizable para otros propósitos. Se usa pues para referirse a algo que es adecuado sólo para un determinado fin. En sentido amplio, ad hoc puede traducirse como «específico» o «específicamente».

- **Backbone:** Estructura de transmisión de datos de una red o conjunto de ellas en Internet. Literalmente: "columna vertebral".

- **CPI:** Las metodologías para la mejora continua de los procesos (CPI, Continuous Process Improvement) como Six Sigma y Lean son una parte natural de BPM. Estos enfoques de eficacia comprobada para la optimización de los procesos amplían su fuerza y alcance cuando se combinan con la tecnología BPM. BPM es la plataforma que lleva CPI al nivel de la empresa.

- **DTD:** Siglas en inglés de Document Type Definition. La Definición de Tipo de Documento (DTD) es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD. De esta forma, dichos documentos, pueden ser validados, conocen la estructura de los elementos y la descripción de los datos que trae consigo cada documento, y pueden además compartir la misma descripción y forma de validación dentro de un grupo de trabajo que usa el mismo tipo de información.

- **Fase:** La fase indica la situación instantánea en el ciclo, de una magnitud que varia cíclicamente.

- **Interoperabilidad:** Es la condición mediante la cual sistemas heterogéneos pueden intercambiar procesos o datos.

- **Metodología:** Metodología, del griego (metà "más allá" odòs "camino" logos "estudio"). Se refiere a los métodos de investigación que se siguen para alcanzar una gama de objetivos en una ciencia. En resumen es el conjunto de métodos que se rigen en una investigación científica o en una exposición doctrinal.

- **TI:** El acrónimo de Tecnologías de la información.

- **time2market:** Time2Market se especializa en el desarrollo y despliegue de soluciones de tecnología avanzada que transformará la tecnología de vanguardia y las mejores prácticas operacionales en sistemas de negocio necesarios para el mundo conectado de hoy.

- **Servicios:** Son actividades de naturaleza frecuentemente inmaterial que sirven, al igual que los bienes para satisfacer los deseos o necesidades del ser humano.

- **Validación:** Confirmación que se da por la recopilación y análisis de la evidencia objetiva de que se cumplen los requisitos particulares para el uso específico propuesto.

- **WSDL:** Son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web.

ANEXOS.

Anexo 1: Encuesta de autovaloración de los expertos.

Compañero (a):

En la ejecución de la presente tesis, deseamos someter a la valoración de un grupo de expertos, la propuesta del Modelo de Prueba para una Arquitectura SOA. Para ello necesitamos conocer el grado de dominio que Ud. posee sobre modelos de prueba; y con ese fin deseamos que responda lo que se le pide a continuación.

Nombre y apellidos: _____

Centro de trabajo: _____

Labor que realiza: _____

Especialidad: _____ Años de experiencia: _____

Categoría docente: _____Categoría científica: _____

1.- Marque con una cruz (X) el grado de conocimiento que Ud. tiene sobre la temática que se investiga:

0	1	2	3	4	5	6	7	8	9	10

2.- Marque con una cruz (X) las fuentes que le han servido para argumentar el conocimiento que tiene Ud. de la temática que se investiga.

No.	Fuentes de argumentación	Grado de influencia		
		Alto	Medio	Bajo
1.-	Análisis realizado por Ud.			

2.-	Experiencia.			
3.-	Trabajos de autores nacionales.			
4.-	Trabajos de autores extranjeros.			
5.-	Su propio conocimiento del tema.			
6.-	Su intuición.			

Anexo 2: Encuesta a los expertos.

Compañero (a):

La presente tesis se propone definir un modelo de prueba para una arquitectura SOA. El modelo muestra la metodología a usar durante el desarrollo de las pruebas, los entregables que recogerán todos los datos de cada fase y los roles involucrados.

A continuación se relacionan un conjunto de afirmaciones que resumen los aspectos más relevantes del modelo propuesto. Valore el grado de factibilidad de los mismos de acuerdo a la siguiente escala.

MA-Muy Adecuado

BA-Bastante Adecuado

A-Adecuado

PA-Poco Adecuado

NA-No adecuado

En todos los casos marque con una (X) su selección:

No	**El modelo V es una adecuada metodología de prueba para entregar proyectos SOA por las siguientes razones:**	**MA**	**BA**	**A**	**PA**	**NA**
1	Estimula una metodología descendente con respecto a la definición de los requisitos del proceso de negocio, de alto nivel del diseño técnico funcional, de seguridad, etc.					
2	Estimula un método de prueba ascendente.					
3	En SOA los servicios son ligeramente acoplados y es por eso que un modelo de pruebas ascendente es recomendable.					
4	Los niveles reflejan diferentes puntos de vista de las pruebas en los diferentes niveles de detalles.					
5	El modelo V estimula las pruebas a lo largo de todo el ciclo de vida de desarrollo del software.					
No	**Las pruebas en una arquitectura SOA se definen en las siguientes fases:**	**MA**	**BA**	**A**	**PA**	**NA**
6	Nivel de prueba de Componentes de Servicios.					
7	Nivel de prueba de Servicios.					

8	Nivel de prueba de Integración.					
9	Nivel de prueba de Proceso/Orquestación.					
10	Nivel de prueba de Sistema.					
	Si lo considera necesario proponga otra(s) fase(es).	*				
No	**Los roles involucrados para la realización de las pruebas en un ambiente SOA son:**	**MA**	**BA**	**A**	**PA**	**NA**
11	Probador.					
12	Analista de pruebas.					
13	Diseñador de Pruebas.					
14	Jefe de Proyecto de Pruebas.					
15	Consultor de pruebas.					
16	Especialista en automatización de pruebas.					
17	Gestor de Calidad.					
	Si lo considera necesario proponga otro(s) rol(es).	*				
No	**Los entregables que se proponen son los siguientes:**	**MA**	**BA**	**A**	**PA**	**NA**
18	Entregable Plan de Pruebas.					
19	Entregable Caso de Pruebas.					

20	Entregable No Conformidades.					
21	Entregable Prueba de Aceptación de Usuario.					
22	Entregable Prueba del Sistema.					
23	Entregable Prueba de Integración.					
	Si lo considera necesario proponga otro(s) entregable(es).	*****				
No	**Los indicadores para la selección de herramientas de pruebas son los siguientes:**	**MA**	**BA**	**A**	**PA**	**NA**
24	Alerta de las interfaces de prueba.					
25	Automatización de pruebas basadas en mensajes.					
26	Virtualización.					
27	Simulación.					
28	Prueba de carga.					
29	Validación.					
30	Componentes.					
31	Introspección.					
32	Administración.					
33	Seguridad.					
34	Formato de datos específicos.					

35	Prueba continua.					
	Si lo considera necesario proponga otro(s) indicador(es).	*				

Anexo 3: Entregable Caso de Prueba.

Diseño de Casos de Prueba

<nombre del proyecto>

<nombre del producto>

<versión>

<nombre del CU>

Control de versiones:

Fecha	Versión	Descripción	Autor
<dd/mm/aa>	<x.x>	<detalles>	<nombre>

Reglas de Confidencialidad

Clasificación: <<Clasificación>>

Este documento contiene información propietaria de**"<<Centro de Consultoría Tecnológica y de Integración de Sistemas>>"** y/o **"<<Empresa Cliente>>"** , y es emitido confidencialmente para un propósito específico.

El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimientos público su contenido, excepto para cumplir el propósito para el cual se ha generado.

Estas reglas son aplicables a las 8 páginas de este documento.

Tabla de Contenido

Descripción General

[*Descripción general del CU.*]

Condiciones de Ejecución:

[*Precondiciones del CU.*]

1 Secciones a probar en el caso de uso:

[*Para cada sección los escenarios van a ser flujo básico + los alternativos.*]

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
[*SC 1: Nombre de la sección*]	*EC 1.1: Nombre del Escenario.*	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 1.2: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 1.n: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
[*SC 2: Nombre de la sección*]	*EC 2.1: Nombre del Escenario.*	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 2.2: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*

	EC 2.n: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
[*SC n: Nombre de la sección*]	*EC 3.1: Nombre del Escenario.*	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 3.2: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 3.n: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*

1.1 SC 1: <Sección #1>

PLANTILLA DE CASO DE PRUEBA	
Introducción/visión general	
<identificador>	[*Es un identificador único para futuras referencias, por ejemplo, mientras se describe un defecto encontrado*]
<nombre>	[E*l caso de prueba debe ser un título entendible por personas, para la fácil comprensión del propósito del caso de prueba y su campo de aplicación*]
<identificador de requerimientos>	[E*stá incluido por el caso de prueba. También*

	aquí puede ser identificador de casos de uso o especificación funcional]
<propósito>	[*Contiene una breve descripción del propósito de la prueba, y la funcionalidad que chequea*]
<dependencias>	[Describir si para la ejecución del caso de prueba existe alguna dependencia específica]
Actividad	
<ambiente de prueba/configuración>	[*Contiene información acerca del la configuración del hardware o software en el cuál se ejecutará el caso de prueba*]
<inicialización>	[*Describe acciones, que deben ser ejecutadas antes de que los casos de prueba se hayan inicializado. Por ejemplo, debemos abrir algún archivo*]
<finalización>	[*Describe acciones, que deben ser ejecutadas después de realizado el caso de prueba. Por ejemplo si el caso de prueba estropea la base de datos, el analista debe restaurarla antes de que otro caso de prueba sea ejecutado*]
<acciones>	[*Pasos a realizar para completar la prueba*]
<descripción de los datos de entrada>	[*Descripción de los datos que funcionan como entrada en el caso de prueba*]

Resultados	
<resultados esperados>	*[Contiene una descripción de lo que el analista debería ver tras haber completado todos los pasos de la prueba]*
<resultados reales>	*[Contienen una breve descripción de lo que el analista encuentra después de que los pasos de prueba se hayan completado. Esto se sustituye a menudo con un Correcto/Fallido. Si un caso de prueba falla, frecuentemente la referencia al defecto implicado se debe enumerar en esta columna]*

2 Registro de defectos y dificultades detectados.

Elemento	**No**	**No conformidad**	**Aspecto correspondiente**	**Etapa de detección**	**Clasificación**	**Estado NC**	**Respuesta del Equipo Desarrollo**
<Nombre del Elemento>	*< 1>*	*<Descripción de la No Conformidad>*	*<Descripción de Aspecto correspondiente>*	*<Etapa de detección del error>*	*S: Significativa* *NS: No Significativa*	*[Se coloca el estado de la NC y la fecha, cada vez que se revise se deja el estado*	*[Esta columna se comienza a llenar a partir de la 2da iteración, y es responsabilidad del equipo de desarrollo,*

					R: Recomendación	*anterior y se coloca el nuevo con la fecha en que se revisó.] RA: Resuelta PD:Pendiente NP:No Procede*	*quien especifica la conformidad con lo encontrado o no y en caso de no proceder la no conformidad explica por qué.]*

3 Anexos

3.1 Anexo <1>

Anexo 4: Entregable Plan de Pruebas.

Plan de Pruebas

<Nombre del Proyecto>

<Nombre del producto>

<Versión>

Control de versiones

Fecha	Versión	Descripción	Autor
<dd/mm/aa>	<x.x>	<detalles>	<nombre>

Reglas de Confidencialidad

Clasificación: <<Clasificación>>

Este documento contiene información propietaria de**"<<Centro de Consultoría Tecnológica y de Integración de Sistemas>>"** y/o **"<<Empresa Cliente>>"** , y es emitido confidencialmente para un propósito específico.

El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimientos público su contenido, excepto para cumplir el propósito para el cual se ha generado.

Estas reglas son aplicables a las 10 páginas de este documento.

Tabla de contenidos

Plan de Pruebas

1 Introduccción

1.1 Alcance

[Proyectos con los que se involucra el Plan]

1.2 Definiciones, Acrónimos y Abreviaturas

1.3 Referencias

[Lista de documentos a los que se hace referencia en el Plan]

Código	Título
[1]	Documento 1
[2]	Documento 2
[n]	…

2 Organización del equipo de pruebas

[Descripción del equipo de probadores, por quienes está compuesto, responsabilidad de cada miembro.]

3 Arquitectura técnica.

[Descripción mediante diagramas de las partes que componen el sistema bajo prueba. Incluye el almacenamiento de datos y las conexiones para su transferencia y describe el objetivo de cada componente, inclusive la forma de su actualización. Se debe documentar tanto las capas (Process Services, Capability Services, Core Business Services, Underlying Services, Utility Services), como la presentación / interfaz del usuario, la base de datos, los emisores de informes, etc. Un diagrama de alto nivel que muestre como el sistema en prueba se inserta en un contexto de automatización mayor también puede ser agregado, si el mismo está disponible.]

4 Especificaciones del Software y Hardware

[Corresponde a una lista individualizada de todo el hardware y el software que utiliza el sistema, incluyendo proveedores y versiones.]

5 Descripción del Plan de Pruebas

5.1 Descripción de los requerimientos

[Esta sección del Plan de Pruebas contiene una lista de todos los requerimientos. Cualquier requerimiento no incluido en esta lista estará fuera del alcance de las pruebas. Esto libera de responsabilidad en caso de que existan problemas con la funcionalidad del sistema que se relaciona con un requisito no incluido en este listado.]

5.1.1 Requerimientos Funcionales

[Todas las funciones que deben ser probadas, como por ejemplo la creación, la corrección y supresión de registros, son puestas en esta lista. Puede incluirse la lista completa en esta sección o bien hacerse referencia a otro documento que contenga la información.]

5.1.2 Casos de Uso: <Caso de Uso1>

[Aquí se debe indicar brevemente en qué consiste el CU, y en las secciones posteriores incluir la información necesaria para verificar su funcionamiento.]

Escenarios

[Aquí se plantea la matriz de escenarios correspondiente al Caso de Uso, esto es todas las posibles combinaciones del flujo normal de los eventos y los flujos alternativos, caminos que se derivan del flujo de eventos del Caso de Uso.]

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo	……

Plantilla de condiciones

[Aquí se plantea la planilla de Condiciones, estas serán las condiciones que causan que se ejecute un escenario específico dentro de los posibles escenarios identificados para el Caso de Uso, indicando el resultado esperado de ejercitar ese Escenario-Condición y la referencia a los casos de prueba de la planilla CasosPruebaConDatos.xls para ese CU que se corresponden con ese Escenario-Condición.]

Escenario	**Descripción**	**Condiciones o Elementos** <Las condiciones o elementos requeridos para ejecutar los distintos escenarios>			**Resultado Esperado**
		Condición 1	Condición 2	…	
Esc 1					
Esc 2					
…					

Diagrama de Entidad Relación

[Entidades que intervienen en el caso de uso en cada uno de los estados por los que transitan.]

Casos de prueba

[Aquí se plantean las pruebas para ese ciclo, dado que una prueba encadena un escenario de cada caso de uso, se debe dar la referencia al Escenario del Caso de Uso, la referencia a los datos que prueban ese escenario y las entidades involucradas en cada prueba y sus estados, para el caso de Prueba del ciclo debe darse el resultado esperado.]

PLANTILLA DE CASO DE PRUEBA	
Introducción/visión general	
<identificador>	[Es un identificador único para futuras referencias, por ejemplo, mientras se describe un defecto encontrado]
<caso de prueba dueño/creador>	[Es el nombre del analista o diseñador de pruebas, quien ha desarrollado pruebas o es responsable de su desarrollo]
<versión>	[Actual definición del caso de prueba]
<nombre>	[El caso de prueba debe ser un título entendible por personas, para la fácil comprensión del propósito del caso de prueba y su campo de aplicación]
<identificador de requerimientos>	[Está incluido por el caso de prueba. También aquí puede ser identificador de casos de uso o especificación funcional]
<propósito>	[Contiene una breve descripción del propósito de la prueba, y la funcionalidad que chequea]

<dependencias>	[Describir si para la ejecución del caso de prueba existe alguna dependencia específica]
Actividad	
<ambiente de prueba/configuración>	[Contiene información acerca del la configuración del hardware o software en el cuál se ejecutará el caso de prueba]
<inicialización>	[Describe acciones, que deben ser ejecutadas antes de que los casos de prueba se hayan inicializado. Por ejemplo, debemos abrir algún archivo]
<finalización>	[Describe acciones, que deben ser ejecutadas después de realizado el caso de prueba. Por ejemplo si el caso de prueba estropea la base de datos, el analista debe restaurarla antes de que otro caso de prueba sea ejecutado]
<acciones>	[Pasos a realizar para completar la prueba]
<descripción de los datos de entrada>	[Descripción de los datos que funcionan como entrada en el caso de prueba]
Resultados	
<resultados esperados>	[Contiene una descripción de lo que el analista debería ver tras haber completado todos los pasos de la prueba]

<resultados reales>	[Contienen una breve descripción de lo que el analista encuentra después de que los pasos de prueba se hayan completado. Esto se sustituye a menudo con un Correcto/Fallido. Si un caso de prueba falla, frecuentemente la referencia al defecto implicado se debe enumerar en esta columna]

5.2 Requerimientos de Diseño

[Las pruebas de la interfaz de usuario, las estructuras de menú u otros elementos de diseño también deberían ser puestas en una lista o referenciados hacia otro documento.]

5.3 Requerimientos de Integración

[Los requerimientos para probar el flujo de datos desde un componente a otro deben ser incluidos si ellos harán parte del Plan de Pruebas.]

5.4 Otros Requerimientos

[Cualesquiera otras exigencias que tenga la aplicación y que necesiten ser probadas.]

6 Estrategia de Prueba

[Use esta sección para describir como los objetivos de la prueba serán alcanzados para cada uno de los tipos de pruebas que hacen parte del plan, ejemplo:

- Unitarias
- De integración
- De sistema
- Validación y Verificación
- De estrés
- De configuración y/o de instalación
- …

Se puede agregar alguna otra prueba que se haga…
Para cada subconjunto requerido o definido como necesario, debe detallarse sus objetivos y lineamientos.]

6.1 Objetivo

[El objetivo global de esta estrategia debe alcanzarse. Por ejemplo, para una prueba de sistema, este objetivo puede ser una declaración de que todos los requerimientos funcionales deben comportarse de acuerdo a lo esperado, o como quedó documentado.]

6.2 Técnica

[Especifica como los casos de prueba serán desarrollados, el instrumento o herramienta usado para almacenarlos y donde pueden ser encontrados; como ellos serán ejecutados y los datos que serán usados. Declare aquí si las pruebas deben ser realizadas en ciclos, o de común acuerdo con los otros esfuerzos de pruebas.]

6.4 Entorno de Prueba

[Especificar las condiciones de hardware y configuración bajo las cuales se deben realizar las pruebas.]

6.5 Proceso

[Breve descripción del proceso que se realiza.]

6.6 Casos de Prueba

[Hacer una lista detallada o una referencia a los casos reales de prueba que serán utilizados para poner en práctica el plan.]

6.7 Criterios de Término

[Registrar los criterios que serán usados para determinar la aprobación o rechazo de pruebas y la acción que debe ser tomada con base en los resultados de la prueba.]

6.8 Herramientas

[Documentar los instrumentos o herramientas que serán empleados para las pruebas.]

7 Recursos Requeridos

[Identificar los roles y las responsabilidades que serán requeridas para la ejecución del Plan de Pruebas, así como los datos de la infraestructura necesaria para realizar el mismo.]

8 Plan de proyecto

[Parte del cronograma del proyecto que abarca la etapa de pruebas.]

9 Calendario y Plazos.

[Documentar el plazo en el cual el sistema a probar estará disponible para pruebas y el tiempo estimado para ejecutar los casos de prueba.]

10 Definición de los Entregables.

[Registrar en una lista cualquier entregable asociado con el esfuerzo de pruebas y donde las copias de estos entregables o documentos pueden ser localizados. Esto incluye el Plan de Pruebas en sí mismo, escenarios para prueba, casos de prueba y el plan de proyecto.]

11 Seguimiento y Reporte de Defectos.

[Documente el instrumento y el proceso usado para registrar y rastrear los defectos. Ponga en una lista todos los informes que serán generados incluyendo repositorios, frecuencias, mecanismos de entrega y ejemplos. Identifique los recursos involucrados en el proceso de seguimiento.

Describa cualesquiera calificación, categoría o clasificación que se usará para identificar o priorizar defectos. Las siguientes son categorías, de ejemplo, para priorizar o calificar defectos:

- **Crítico**: Denota una función inutilizable que causa un término anormal o una falla general, o cuando un cambio en un área de la aplicación causa un problema en otra parte.
- **Severo**: Una función no actúa como fue requerido o diseñado, o un objeto de interfaz no trabaja como se muestra.
- **Advertencia**: La función trabaja, pero no tan rápidamente como esperado, o no se ajusta a las normas y convenciones.
- **Cosmético**: No crítico para el funcionamiento de sistema: palabras con mala ortografía, formateo incorrecto, mensajes de error vagos o confusos o advertencias.]

12 Aprobación del Plan.

[El Plan de Pruebas debe ser revisado por todos las partes responsables de su ejecución y aprobado por el equipo de prueba, el jefe del proyecto y el gerente de desarrollo. Obtenga las firmas de aprobación en todas las páginas del mismo.

Una reunión final de verificación con todas las partes involucradas, es comprobadamente el método más eficaz para obtener la aprobación del Plan de Pruebas.]

13 Documentación de los Resultados.

[Cuando el esfuerzo de prueba esté terminado, documente los resultados y mediciones. Identifique cualquier discrepancia entre el plan y la puesta en práctica real y describa adecuadamente como aquellas discrepancias fueron manejadas.]

Caso de Prueba	
Fecha	
Resultado	
Observaciones	
Responsables de las Pruebas	

Anexo 5: Entregable No Conformidades.

NO CONFORMIDADES

<Nombre del proyecto>

<Nombre del producto>

<Versión>

Control de versions

Fecha	Versión	Descripción	Autor
<dd/mm/aa>	<x.x>	<detalles>	<nombre>

Reglas de Confidencialidad

Clasificación: <<Clasificación>>

Este documento contiene información propietaria del**"<<Centro de Consultoría Tecnológica y de Integración de Sistemas>>"** y/o **"<<Empresa Cliente>>"** , y es emitido confidencialmente para un propósito específico.

El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimientos público su contenido, excepto para cumplir el propósito para el cual se ha generado.

Estas reglas son aplicables a las 4 páginas de este documento.

1 Descripción General

[*Descripción de Aspectos Generales a tener en cuenta a la hora de realizar el diseño de las pruebas, incidencias en el momento de su desarrollo y otros aspectos relevantes*]

2 Elementos probados

[*Descripción general o lista de los Elementos Probados, y otros aspectos importantes a tener en cuenta a la hora analizar las No Conformidades Detectadas*]

3 Elementos no probados y causas

[*Descripción de Aspectos Generales a tener en cuenta a la hora de realizar el diseño de las pruebas, incidencias en el momento de su desarrollo y otros aspectos relevantes*]

4 Registro de defectos y dificultades detectados

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Clasificación	Estado NC	Respuesta del Equipo Desarrollo
[*Nombre del Elemento*]	*[1]*	[*Descripción de la No Conformidad*]	[*Descripción del Aspecto correspondiente*]	[*Etapa de detección del error*]	[*S: Significativa*] [*NS: No Significativa*] [*R: Recomendación*]	[*Se coloca el estado de la NC y la fecha, cada vez que se revise se deja el estado anterior y se coloca el nuevo con la fecha en que se revisó*] ***RA**: Resuelta* ***PD**: Pendiente* ***NP**: No Procede*	[*Esta columna se comienza a llenar a partir de la 2da iteración, y es responsabilidad del equipo de desarrollo, quien especifica la conformidad con lo encontrado o no y en caso de no proceder la no conformidad explica por qué*]

[*La NC puede tener solo una de las tres clasificaciones: Significativa, No Significativa o Recomendación*]

5 Anexos