Universidad de las Ciencias Informáticas.

Facultad 5

Título: Guía de trabajo para las prácticas Diseño Incremental y Pruebas Continuas de la metodología XP para el diseño de aplicaciones compuestas.

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autora: Dimelza del Pilar Remedios Cruz.

Tutor: Ing. Orestes Febles Díaz.

Co-Tutora: Ing. Maité Rivero Cruz.

La Habana, 2012.

"La disciplina quiere decir orden, y el orden quiere decir triunfo"

José Martí.

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Dimelza del Pilar Remedios Cruz **Orestes Febles Díaz**

_____ _____

Firma del Autor Firma del Tutor

Agradecimientos

Cuando hay que agradecer a las personas que te ayudaron con tu tesis vienen muchos nombres a la mente, pero hay algunos que no solo aparecen en ese momento, sino que siempre están presentes, es a esas personas a las que agradezco.

A mi tutor Orestes Febles.
A mi co-tutora la Ing. Maité Rivero Cruz.
A la Ing. Jessie Castell.
Al Ing. Ariel Viera.
Al Profesor Manuel Villanueva Betancourt.

Dedicatoria

Quisiera dedicar esta tesis a todas las personas que son realmente importantes para mí, personas a las que no imagino lejos de mí y que han sido un factor fundamental en la creación de esta tesis.

A mi madre por tenerme paciencia y fe.

A mi novio por ayudarme y darme fuerzas cuando pensé que no podía

A mi papá por guiarme, enseñarme y aconsejarme.

A mis suegros por apoyarme en todo momento.

Resumen

En el presente trabajo se realizó un estudio de la metodología ágil de desarrollo de software Programación Extrema. Como parte de este estudio se analizaron sus fases, sus roles, sus prácticas y otros elementos, con el objetivo de determinar qué fase puede tener mayor influencia en el proceso de desarrollo de las aplicaciones compuestas; lo cual permitió elegir las prácticas de la fase seleccionada más importantes para el desarrollo de este tipo de aplicaciones. Este estudio sirvió de base para la creación de una guía de trabajo sobre el funcionamiento de las prácticas seleccionadas, modificándose su proceso de ejecución en función del diseño de aplicaciones compuestas específicamente, la cual es utilizable en cualquier proyecto que desarrolle estas aplicaciones. Por lo que en la guía se muestra cual es el papel de cada una de las prácticas escogidas en el proceso de diseño de una aplicación compuesta, además de especificarse cuáles son las actividades que se realizan en cada una de ellas, sus roles y artefactos. Por último se realizó la validación de dicha guía por dos vías, mediante la realización de varios casos de estudio para validar funcionalmente la guía y a través del Método de Expertos Delphi para validar su calidad.

Palabras clave

Aplicaciones compuestas, Metodología ágil, Programación Extrema.

Índice

Introducción

El surgimiento de Internet fue uno de los grandes cambios que sucedieron en el mundo de la informática a partir de 1980, en ese momento el mercado informático se volvió más competitivo, grandes compañías de software como Microsoft e IBM ampliaron su área de producción para poder suplir las exigencias de los clientes[1] que se volvían cada vez más crecientes, a medida que se generalizaba el uso de la informática en la sociedad. Además la economía mundial sufrió un cambio radical que repercutió grandemente en los mercados informáticos, ya que pasó de ser una economía basada en los bienes, a una basada en los servicios. De modo que ganaron en importancia los servicios que se pudieran obtener a partir de la WWW[1] [1, 2].

En la década del 90 surge las aplicaciones compuestas[1]. Una aplicación compuesta es un programa informático que agrupa un conjunto de activos de software que se han reunido para ofrecer una capacidad de negocio. Estos activos son artefactos que se pueden implementar de manera independiente, permitiendo la composición y el aprovechamiento de las capacidades específicas de la plataforma[3]. Además su composición permite que sus componentes se descubran y combinen en tiempo de ejecución de la aplicación [4]. De manera que cualquier modificación que sea necesario realizarle a cualquiera de ellos puede hacerse sin que tenga repercusiones negativas sobre los demás, permitiendo también su actualización de manera dinámica.

A nivel mundial el mercado de demanda de aplicaciones compuestas es cada vez más grande y profesional[5]. Esto se debe a que estas aplicaciones brindan una serie de beneficios a sus usuarios; tales como agilidad para el negocio que las utilice, adaptabilidad ante los cambios y alineamiento entre el personal del área de negocios y el del área de IT[2] [3].Por lo cual el desarrollo de este tipo de aplicaciones se ha vuelto una necesidad para los desarrolladores de software que abastecen o desean abastecer este mercado[1].

[1] World Wide Web, servicio de Internet que permite acceder a millones de páginas web que contienen toda clase de documentos e información.
[2] Se refiere al área de Tecnologías de la Información.

Sin embargo para realizar este tipo de aplicaciones con la calidad requerida es necesario utilizar un elemento fundamental en un proceso de desarrollo de software: una metodología de desarrollo de software. Este elemento es una filosofía de desarrollo de programas de computación acompañada de herramientas, modelos y métodos, que es usada para estructurar, planear y controlar el proceso de desarrollo de un software[6].

El desarrollo de las metodologías se originó con el objetivo de poder desarrollar a gran escala y de forma deliberada, estructurada y metódica los sistemas de negocio [7]. Las primeras metodologías aparecen en la década de los 80 y son conocidas por su forma de desarrollo como metodologías tradicionales, un ejemplo de estas es la Metodología de Análisis y Diseño Estructurado de Sistemas (del inglés Structured Systems Analysis and Design Methodology) [8]. A partir de la década del 90 se inicia el desarrollo de metodologías con una filosofía de desarrollo diferente a las vistas hasta ese momento, las cuales se conocen como metodologías ágiles. Este tipo de metodología rompe casi por completo con el esquema de desarrollo de software planteado por las anteriores; proponiendo un proceso de desarrollo donde el bien principal son los recursos humanos. Su mayor exponente en la actualidad es la metodología Programación Extrema del ingles eXtreme Programming (XP)[9].

Con el objetivo de mejorar su proceso de ejecución las metodologías presentan un conjunto de técnicas de modelado de sistemas, incluyendo heurísticas de construcción y criterios de comparación de modelos de sistemas. Integrando estas técnicas dentro de una serie de fases que rigen el proceso de desarrollo, dividiéndolo en etapas. Etapas que se conocen en las metodologías tradicionales como Flujos de trabajo y en las ágiles se les da el nombre de Prácticas. Estas prácticas han sido clasificadas en cuanto a los roles que las desempeñan y en cuanto a las etapas del proceso de desarrollo donde su uso es indispensable, como es el caso de las prácticas Diseño Incremental y Pruebas Continuas que pertenecen a la etapa de diseño del software y son prácticas del rol programador [9, 10].

Ambas tendencias metodológicas, tradicionales o ágiles tienen sus pros y sus contras. Por esta razón es necesario que en el proceso de selección de una metodología para el desarrollo de un software se tenga en cuenta también cuáles son las características específicas de ese software, de manera que se escoja la metodología correcta para su desarrollo en particular. Teniendo en cuenta que las aplicaciones

compuestas aportan agilidad a la respuesta de las organizaciones ante los cambios, se considera que es necesaria la utilización de una metodología que aporte agilidad y rapidez a su proceso de desarrollo, como es el caso de las metodologías ágiles. Además existen escenarios donde es conveniente el uso de este tipo de metodologías para el desarrollo de aplicaciones compuestas, por lo cambiante de los requisitos que rigen su desarrollo o por la necesidad de realizar un proceso de desarrollo de corto plazo[11].

En el Centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE), de la Universidad de las Ciencias Informáticas (UCI), se han desarrollado varios proyectos de desarrollo de aplicaciones compuestas, como es el caso del proyecto “Gestión de Tesis”. Esto se debe a que este centro ofrece soluciones estrechamente relacionadas con soluciones SOA (Arquitectura Orientada a Servicios) como la prestación de servicios y estas aplicaciones cumplen con el modelo de prestación de servicios, además de estimular la reutilización de código y la rápida respuesta a los cambios durante el proceso de desarrollo. Pero no solo es necesario realizar estas aplicaciones, sino que es necesario realizarlas de manera ágil y rápida, lo cual puede lograrse mediante el uso de una metodología ágil como XP, para guiar su proceso de desarrollo [12]. Sin embargo la realización de las aplicaciones compuestas presenta problemas ya que su desarrollo no es trivial, debido a que una gran parte de sus funcionalidades son compuestas fuera de sus fronteras. Lo que produce problemas en particular durante la etapa de diseño[11]. Entre las principales dificultades se encuentran:

- No existe un alto grado de confiablidad en la composición de servicios que se prestan en las aplicaciones, en cuanto al factor de verificación y validación de dichos servicios, ya que estos aspectos no se realizan con el nivel de profundidad necesario.
- No se realiza una planificación adecuada del desarrollo de las aplicaciones, en específico de la etapa de diseño.
- Se presentan errores en etapas finales del desarrollo de la aplicación, por la no comprobación de las funcionalidades con anterioridad, lo cual conlleva gastos en recursos y tiempo.

Y aunque existen proyectos en los que se está utilizando la metodología XP para el proceso de desarrollo de estas aplicaciones[13], la bibliografía relacionada a esta etapa no está accesible para el público.

Debido a la **situación problémica** antes expuesta, se plantea el siguiente **problema de investigación**:

¿Cómo ganar en agilidad y organización en la etapa de diseño durante el proceso de desarrollo de aplicaciones compuestas a través del uso de la metodología XP?

El desarrollo de este trabajo tendrá como **objeto de estudio:** Las fases de la metodología ágil XP, donde el **objetivo de la investigación** es: Crear una guía de trabajo sobre las prácticas Diseño Incremental y Pruebas Continuas de la metodología XP para el desarrollo de aplicaciones compuestas. La investigación se centrará fundamentalmente en el **campo de acción:** Las prácticas Diseño Incremental y Pruebas Continuas de la metodología XP para el diseño de aplicaciones compuestas.

Para lograr el objetivo general de la investigación se trazaron los siguientes **objetivos específicos**:

- Elaborar un marco teórico referencial relacionado con el objeto de estudio.
- Realizar un diagnóstico de la situación actual en cuanto a la utilización de la metodología XP para el desarrollo de aplicaciones compuestas.
- Elaborar una propuesta de guía de trabajo a partir del estudio del marco teórico sobre el funcionamiento de las prácticas Diseño Incremental y Pruebas Continuas, para la etapa de diseño de las aplicaciones compuestas.
- Realizar la validación de la guía a través de métodos matemáticos.

Para complementar la realización de los objetivos específicos trazados, se programaron las **tareas de investigación** siguientes:

- Elaboración del marco teórico de la investigación a partir del estado del arte existente actualmente sobre el tema.
- Realización de un análisis del funcionamiento de la metodología ágil XP para conocer la función de las prácticas dentro del mismo.
- Elaboración de un estudio sobre el funcionamiento de las prácticas Diseño Incremental y Pruebas Continuas dentro de la metodología XP, para realizar una guía de trabajo sobre estas prácticas.
- Selección de los roles que formarán parte de la guía de trabajo.
- Planificación de los pasos que conformarán la guía de trabajo.
- Selección de los artefactos que se utilizarán y generarán durante la ejecución de la guía de trabajo.
- Selección de la vía (teórica, práctica, o ambas) a seguir para la validación de la guía de trabajo.

- Realización de entrevistas a especialistas, para conocer su opinión sobre el tema.
- Realización de encuestas a especialistas, para conocer su nivel de conocimiento sobre el tema y su opinión sobre la guía de trabajo.
- Implantación de la guía de trabajo en los casos de estudio.

La **Idea a defender** en esta investigación es la siguiente: La aplicación de la guía de trabajo de las prácticas Diseño Incremental y Pruebas Continuas permitirá ganar en agilidad y organización en la etapa de diseño del proceso de desarrollo de las aplicaciones compuestas.

En la realización de la presente investigación se utilizaron varios **Métodos de investigación,** teóricos empíricos y matemáticos.

Entre los **Métodos teóricos** se encuentran:

El método **Análisis y Síntesis** que permitió llegar a conclusiones, a partir del estudio realizado del tema objeto de investigación, en diferentes fuentes utilizadas para procesar la información obtenida.

El método **Histórico y Lógico** a través del cual la autora pudo conocer el comportamiento y evolución del objeto de estudio.

El método **Sistémico** que fue fundamental para lograr un análisis iterativo e incremental del objeto de estudio.

Los **Métodos empíricos** utilizados fueron:

La **Entrevista abierta** con al que se logró recopilar información sobre el tema a través de personal conocedor del mismo.

El **Análisis de documentos** mediante la consulta de la literatura especializada en el tema rector del trabajo de diploma, para extraer la información necesaria que permiten realizar el proceso de investigación.

El **Método Matemático** que se utilizó para realizar la validación de la guía de trabajo fue:

El **Método de Selección de Expertos (Método Delphi)** que se utilizó para validar la calidad de la guía de trabajo.

El **Valor del resultado obtenido** que este trabajo de diploma brinda es un valor metodológico. El resultado del mismo se centrará en una guía para el trabajo sobre las prácticas Diseño Incremental y

Pruebas Continuas, que presenta la metodología XP. Con el objetivo de facilitar el proceso de desarrollo de aplicaciones compuestas, y mejorar el desempeño de los trabajadores implicados en el mismo.

El trabajo de diploma está conformado por tres capítulos. El primero está destinado para realizar un estudio de las cualidades y elementos esenciales de la metodología XP y su utilización para el desarrollo de aplicaciones compuestas. En el segundo capítulo se presentará una propuesta de guía de trabajo sobre las prácticas Diseño Incremental y Pruebas Continuas de dicha metodología, para el desarrollo de aplicaciones compuestas. Y en el tercer capítulo se realizará la validación de la guía de trabajo presentada en el capitulo dos. Por último se exponen las Conclusiones Generales de la tesis, las Recomendaciones, las Referencias Bibliográficas utilizadas, el Glosario de Términos y Siglas y los Anexos.

Capítulo 1: Fundamentación teórica

En este capítulo se sustenta teóricamente la creación de una guía de trabajo sobre las prácticas de la metodología XP Diseño Incremental y Pruebas Continuas para el diseño de aplicaciones compuestas. En tal sentido, las bases teóricas se refieren a los siguientes aspectos: elementos de la metodología XP, características y beneficios de las aplicaciones compuestas y vinculación de la metodología antes mencionada para la realización de este tipo de aplicaciones.

1.1 Evolución de las metodologías de desarrollo de software.

En el caso del desarrollo de un software una metodología es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en un sistema de información. Además de indicar cómo hay que obtener los distintos productos parciales y finales [14]. Actualmente la metodología es un elemento fundamental en el proceso de desarrollo de un software, sin embargo no siempre fue así. La primera generación de desarrollo de software utilizaba el método de Desarrollo Convencional, sin metodología[15], pero este método ocasionaba una serie de efectos negativos en el proceso de desarrollo como son:

- Los resultados finales son impredecibles.
- No hay forma de controlar lo que está sucediendo en el proyecto.
- Los cambios organizativos afectan negativamente al proceso de desarrollo.

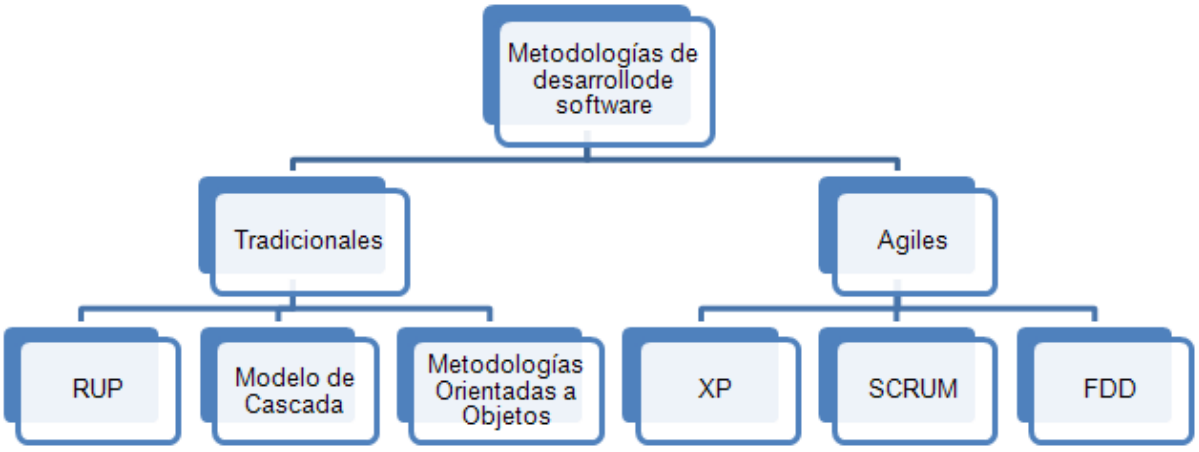

Figura 1. Metodologías de desarrollo de software. Fuente: Elaboración Propia.

Para eliminar estas fallas y debido a que el desarrollo de software no es una tarea fácil, a lo largo de los años se fueron desarrollando diferentes modelos de metodologías. Primero surgió el Desarrollo Estructurado, posteriormente apareció el Desarrollo Orientado a Objetos, cuya esencia es la identificación y organización de conceptos del dominio de la aplicación y no tanto de su representación final en un lenguaje de programación[16].

Las metodologías Orientadas a Objetos y otras surgidas en la misma etapa son conocidas como metodologías Tradicionales, debido a que muestran un proceso de desarrollo centrado especialmente en el control estricto de dicho proceso; estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir y las herramientas y notaciones que se usarán. Además de estar caracterizadas por la abundante presencia de roles y artefactos[16]. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos por la rigidez y complejidad de su proceso de desarrollo [17]. Más tarde con el objetivo de mostrar otra perspectiva del proceso de desarrollo de software aparecen las metodologías ágiles. Estas metodologías se centran más en otras dimensiones de dicho proceso, como el factor humano o el producto en sí. Este enfoque está demostrando su efectividad en proyectos donde los requisitos son muy cambiantes como es el caso de los proyectos de realización de aplicaciones compuestas, o en otros donde se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad [17].

1.2 Metodologías ágiles para el desarrollo de software.

Aunque las metodologías ágiles aparecieron en la década de los 90, es en una reunión celebrada en el año 2001 que nace el término “ágil” aplicado al desarrollo de software. En esta reunión participaron un grupo de expertos de la industria del software, con el objetivo de esbozar los valores y principios, que deberían permitir a los equipos desarrollar software rápidamente y responder a los cambios que puedan surgir a lo largo del proyecto[17]. Estos valores y principios se encuentran resumidos en el documento conocido como “Manifiesto Ágil”, sus principales valores son[17]:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas
- Desarrollar software que funciona más que conseguir una buena documentación.
- La colaboración con el cliente más que la negociación de un contrato.
- Responder a los cambios más que seguir estrictamente un plan.

Por su parte los principios de las metodologías ágiles no son más que las características que diferencian un proceso ágil de uno tradicional. Dichas diferencias se pueden ver de manera resumida en el Tabla 9, que fue extraída del trabajo de Patricio Letelier y María Carmen Peadés, titulado “Metodologías ágiles para el desarrollo de software: Extreme Programming (XP)”.

Para esclarecer la importancia que han tenido las metodologías ágiles en el desarrollo de software solo hay que mirar las estadísticas[7]. Estas muestran que a principios de la década del 90, cuando el uso de metodologías estaba regido principalmente por las metodologías tradicionales, del 100 % de los proyectos de realización de software que se hacían a nivel mundial solo el 16% eran exitosos, mientas que un elevado 53% se calificaban de problemáticos y el 31% restante terminaban en fracaso. Ya para la segunda mitad de la década con las metodologías ágiles empezando a despuntar el porcentaje de éxito en los proyectos aumentó a un 26%, para que los proyectos con problemas disminuyeran a un 43 % y los fracasos a un 28%. Finalmente para el año 2004 con las metodologías ágiles siendo ya reconocidas a nivel mundial el porcentaje de éxito se elevó hasta alcanzar el 29%, mientras que el porciento de fracasos disminuyó a un bajísimo 19%.

1.2.1 Metodología XP.

De las metodologías ágiles, XP es la más popular en la actualidad[12]. XP es una metodología centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Esta metodología se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, en la comunicación fluida entre todos los participantes, la simplicidad en las soluciones implementadas y el coraje para enfrentar los cambios. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Construye un proceso de diseño evolutivo que se basa en mejorar un sistema simple en cada iteración. Todo el diseño se centra en la iteración actual y no se hace nada anticipadamente para necesidades futuras, resultando en un proceso de diseño disciplinado y adaptable de una manera que

indiscutiblemente la hace la más desarrollada de entre todas las metodologías ágiles [18].Además cuenta con un elemento que eleva la eficiencia de su proceso de diseño, las **prácticas**, que vienen a sustituir a los flujos de trabajo de las metodologías tradicionales, y son un conjunto de actividades simples que guían los diferentes aspectos del desarrollo para seguir el proceso, dividiéndose entre las diferentes aéreas del desarrollo. A continuación se presentan las características esenciales de XP organizadas en cuatro apartados: historias de usuario, roles, prácticas y fases.

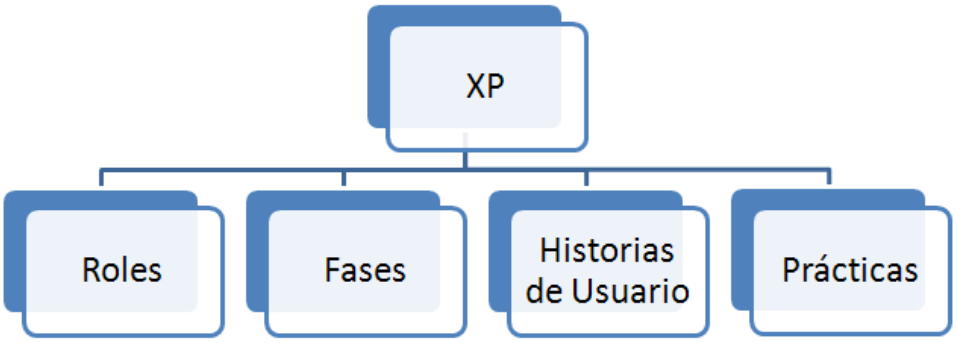

Figura 2. Principales elementos de la metodología XP. Fuente: Elaboración propia.

1.2.1.1 Las Historias de Usuario (HU).

Las historias de usuario son el artefacto utilizado en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Respecto a la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no existe un consenso al respecto. Sin embargo una ya reconocida es una en la que pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado de cosas por terminar y comentarios[12]. Además al comienzo de cada iteración estarán registrados los cambios en las historias de usuario y según eso se planificará la siguiente iteración, así que no hay que preocuparse si en un principio no se identifican todas las historias de usuario[17].

1.2.1.2 Roles.

Siendo XP una metodología que se adapta a las características y necesidades del proyecto, es posible que dependiendo de la situación en la que se aplique se utilice un rol más o uno menos, pero los que se muestran a continuación son los roles que se plantearon es la propuesta original de la metodología.

- **Programador:** El programador escribe las pruebas unitarias y produce el código del sistema. Debe existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo.
- **Cliente:** El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.
- **Probador:** El probador ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Encargado de seguimiento:** El encargado de seguimiento proporciona realimentación al equipo en el proceso XP. Su responsabilidad es verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones. También realiza el seguimiento del progreso de cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos presentes. Determina cuándo es necesario realizar algún cambio para lograr los objetivos de cada iteración.
- **Entrenador:** Es responsable del proceso global. Es necesario que conozca a fondo el proceso XP para proveer guías a los miembros del equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- **Consultor:** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema específico.
- **Jefe:** Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación [17].

1.2.1.3 Prácticas.

Las prácticas que utiliza XP no son específicas de esta metodología, sin embargo es en XP donde se consigue el mayor beneficio de ellas mediante su aplicación conjunta y equilibrada puesto que se apoyan unas en otras, supliendo sus fallas entre si[6]. XP las integra de una forma efectiva y las complementa con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo [12].

El juego de la planificación: En esta práctica el equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración estableciendo la prioridad de cada historia de usuario, de acuerdo con el valor que aporta para el negocio. Los programadores estiman el esfuerzo asociado a cada historia de usuario. Se ordenan las historias de usuario según prioridad y esfuerzo, y se define el contenido de la entrega y/o iteración, apostando por enfrentar lo de más valor y riesgo cuanto antes [19].

Entregas pequeñas: Esta práctica se centra en producir rápidamente versiones del sistema que sean operativas, aunque obviamente no cuenten con toda la funcionalidad pretendida para el sistema, pero que sí constituyan un resultado de valor para el negocio. Una entrega no debería tardar más de 3 meses [20].

Metáfora: En XP no se enfatiza la definición temprana de una arquitectura estable para el sistema. Dicha arquitectura se asume evolutiva y los posibles inconvenientes que se generarían por no contar con ella explícitamente en el comienzo del proyecto se solventan con la existencia de una metáfora. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Donde una metáfora es una historia compartida que describe cómo debería funcionar el sistema [19].

Diseño simple: Esta práctica plantea que se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. La complejidad innecesaria y el código extra debe ser removido inmediatamente [21].

Pruebas: La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del

sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse [22].

Refactorización: La refactorización es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. La refactorización mejora la estructura interna del código sin alterar su comportamiento externo [18].

Programación en parejas: Toda la producción de código debe realizarse con trabajo en parejas de programadores. Puesto que este estilo de programación brinda una amplia gama de beneficios, los principales son: muchos errores son detectados conforme son introducidos en el código, por consiguiente la tasa de errores del producto final es más baja, los diseños son mejores y el tamaño del código menor, los problemas de programación se resuelven más rápido, se posibilita la transferencia de conocimientos de programación entre los miembros del equipo y los programadores conversan, mejorando así el flujo de información y la dinámica del equipo [19].

Propiedad colectiva del código: Esta práctica indica que un programador puede cambiar cualquier parte del código en el momento que lo considere necesario. Esta práctica motiva a todos a contribuir con nuevas ideas en todos los segmentos del sistema, evitando a la vez que algún programador sea imprescindible para realizar cambios en alguna porción de código.

Integración continua: Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día. Todas las pruebas son ejecutadas y tienen que ser aprobadas para que el nuevo código sea incorporado definitivamente.

40 horas por semana: Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse.

Cliente in-situ: El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Gran parte del éxito del proyecto XP se debe a que es el cliente quien conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada.

Estándares de programación: XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación (del equipo, de la organización u otros estándares reconocidos para los lenguajes de programación utilizados). Los estándares de programación mantienen el código legible para los miembros del equipo, facilitando los cambios[17, 19, 21].

1.2.1.4 Fases.

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de vida ideal de XP se divide en seis fases: Exploración, Planificación de la Entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

Fase I: Exploración.

En esta fase los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

Fase II: Planificación de la Entrega.

En esta fase el cliente establece la prioridad de cada historia de usuario, y recíprocamente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses.

Fase III: Iteraciones.

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración. Al final de la última iteración el sistema estará listo para entrar en producción.

Fase IV: Producción.

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento)[17].

Fase V: Mantenimiento.

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

Fase VI: Muerte del Proyecto.

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo [17].

1.2.1.5 Análisis de las fases y prácticas de la metodología XP.

Para darle solución al ciclo de vida antes mencionado, se dividen las prácticas entre las fases, teniendo en cuenta su funcionalidad y objetivos y la funcionalidad que tenga cada fase dentro del desarrollo del software. Lo que no significa que cada práctica solo se utilice en una fase como muestra la siguiente figura.

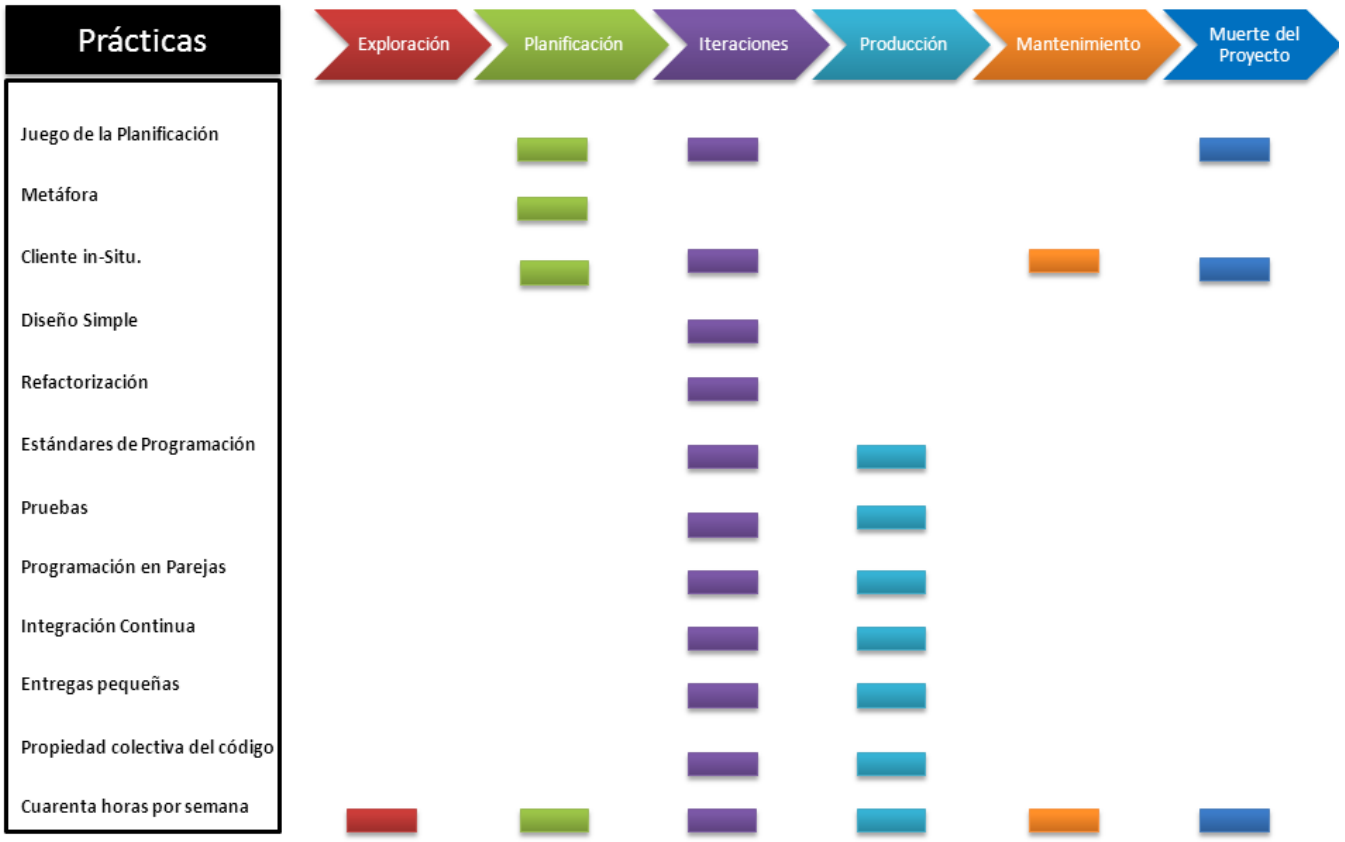

Figura 3. Fases y prácticas que guían el desarrollo de XP. Fuente: Elaboración propia.

- En la fase de Exploración por las características de su funcionamiento solo se utiliza práctica Cuarenta horas por semana. Mientras que los miembros del equipo de desarrollo deciden qué prácticas se deben utilizar en cada una de las siguientes fases del proceso de desarrollo.

- Durante la fase de Planificación con el objetivo de, establecer la prioridad de las HU, realizar una estimación del esfuerzo que conlleva la realización de cada una de ellas y tomar acuerdos relacionados con la primera entrega de la aplicación, se utilizan las prácticas Juego de la Planificación, Metáfora y Cliente in-Situ.

- La fase de Iteraciones es donde más prácticas se utilizan. Esto se debe a que esta es una de las más largas y cargada de trabajo de la metodología, puesto que durante esta fase el diseño del software va a cambiar todas las veces que sea necesario, hasta que se obtenga el diseño deseado por el cliente. De manera que una vez terminada esta fase el software esté en condiciones para empezar su construcción. Por lo que con vista a realizar un diseño de la aplicación con la calidad requerida se utilizan las prácticas: Diseño Simple, Refactorización, Pruebas, Programación en Parejas, Integración Continua, Entregas pequeñas, Cliente In-Situ y Propiedad colectiva del código. Y además en caso de ser necesario se utiliza la práctica Juego de la Planificación para planificar la siguiente iteración.

- Ya en la fase de Producción se procede a implementar la aplicación, lo cual representa bastante trabajo para los programadores. Por lo que las prácticas que se utilizan en esta fase están dirigidas a mejorar el ambiente de trabajo de los programadores y a garantizar la calidad del código. Estas prácticas son: Programación en Parejas, Estándares de Programación, Integración Continua, Pruebas, Entregas Pequeñas y Propiedad colectiva del código.

- En la fase de Mantenimiento por la naturaleza de las actividades que se realizan en ella, destinadas a mantener el sistema en constante funcionamiento mediante la realización de tareas de menor envergadura, mientras que se realiza la producción de la primera versión de la aplicación. Es que en esta fase solo se ejecuta específicamente la práctica Cliente In-situ. Y el resto de las prácticas que se utilicen dependen de las tareas que se realicen para el mantenimiento del sistema, utilizándose las prácticas que faciliten o permitan la realización de dichas tareas.

- La fase de Muerte del proyecto se encuentra en una situación especial ya que en ella no es necesario trabajar más en el desarrollo del proyecto. Debido a que esta fase tiene lugar en el

momento de culminación del proyecto; lo mismo por la satisfacción plena del cliente como por la imposibilidad de continuar con el mismo. Es así que en esta fase se utilizan las prácticas Cliente in-Situ y Juego de la Planificación[17].

- Es importante aclarar que la práctica Cuarenta horas por semana se utiliza en todas las fases del proceso de desarrollo. Esta es una práctica que no debe dejarse de utilizar, pero en caso de que se considere necesaria la utilización de más de cuarenta horas semanales para la realización del proyecto este cambio solo debe realizarse por una semana. Si al cabo de este tiempo el proyecto sigue necesitando más de cuarenta horas semanales para su correcta realización, es necesario analizar las estimaciones realizadas en la fase de Planificación entre otros aspectos para analizar qué medidas tomar, retomándose la utilización de esta práctica[6].

Después de estudiar la división de las prácticas entre las fases del ciclo de vida de XP y haber analizado la literatura existente sobre el funcionamiento de esta metodología, la autora considera que la fase de mayor importancia en el desarrollo de un software es la fase de Iteraciones. Puesto que es en ella donde se realiza el diseño de la aplicación. Es aquí donde se le efectúan a la aplicación los cambios necesarios hasta lograr el diseño deseado por el cliente. Es en esta fase donde se aclaran todos los detalles de cada funcionalidad de la aplicación y se desecha todo lo inservible o innecesario. De forma que cuando esta fase termine y se pase a la producción de la aplicación, los cambios que se le tengan que realizar sean mínimos, puesto que ya el sistema cuenta con todos los requisitos necesarios para su funcionamiento. Además:

- A pesar de la importancia de las fases anteriores, como la de Planificación donde se crean las Historias de Usuario que contendrán los requerimientos de la aplicación, si no se realiza un análisis y diseño adecuado de las mismas en la fase de Iteraciones, la aplicación no tendrá la calidad deseada aun a pesar del buen trabajo hecho durante la Planificación.
- Las fases que le siguen a pesar de ser imprescindibles, dependen de los resultados y del funcionamiento de esta fase para poder tener ellas un funcionamiento con la calidad requerida.

Teniendo en cuenta las actividades a las que está destinada esta fase, y el grado de importancia que tienen las funciones que realizan dentro de ella cada una de las nueve prácticas que utiliza, la autora de

esta investigación considera que de estas las más importantes son las prácticas: Diseño Simple, Refactorización y Pruebas. Esta conclusión está basada en las siguientes razones:

- El objetivo principal de esta fase es crear el diseño de las funcionalidades de la aplicación que se esté desarrollando.
- En la primera de estas tres prácticas se consigue que el diseño de la aplicación cumpla con las directrices de diseño de las metodologías ágiles, además de satisfacer al cliente cumpliendo con sus deseos y expectativas.
- Por su parte la práctica Refactorización logra que el diseño alcance su expresión óptima, mediante su mejora y simplificación.
- Nada de lo anteriormente mencionado fuera posible sin la utilización de la práctica Pruebas. Que permite comprobar la funcionalidad del sistema a medida que se va realizando cualquier modificación, para garantizar la calidad del trabajo que se realiza.
- En el caso de las prácticas Propiedad colectiva del código y Programación en Parejas, estas ayudan a los programadores con el proceso de realización. Pero a pesar de la ayuda que representan, un solo programador también puede hacer este trabajo creando su propio código.
- Las prácticas Integración Continua y Entregas Pequeñas a pesar de ser necesarias no podrían realizarse sin tener de antemano el resultado de las tres prácticas consideradas anteriormente como principales.
- Por su parte la práctica Cliente In-situ tiene un funcionamiento subordinado a la práctica de Pruebas, mientras que la práctica Juego de la Planificación es de uso opcional.

Demostrándose de esta manera el por qué de la elección de las prácticas Diseño Simple, Refactorización y Pruebas, como las prácticas principales de la fase de Iteraciones. Y las que más pueden afectar el proceso de diseño de las aplicaciones compuestas.

Sin embargo la metodología XP desde su primera aparición en el año 1999 ha ido evolucionando. En su última versión, sacada a la luz por su creador Kent Beck en el año 2005, el número de prácticas y sus funcionamientos habían sufrido algunas variaciones. Desapareciendo completamente algunas o descomponiéndose en prácticas que abarcan procesos más pequeños y apareciendo otras nuevas. Para formar un total de 24 prácticas de las 12 que tenía la versión inicial [10]. Debido a este cambio, con el

objetivo de brindar una propuesta actual se decidió trabajar para la guía con las prácticas que se presentaban en la nueva versión.

Con este objetivo se realizó un estudio de las prácticas que intervienen en la etapa de diseño de la nueva versión. Tras estudiar su funcionamiento se concluyó que las prácticas que en el análisis anterior se habían considerado como las prácticas principales por su funcionamiento y la fase donde se ubicaban, no habían sufrido grandes variaciones durante la evolución de la metodología. Sino que se habían fusionado, para formar las prácticas Diseño Incremental y Pruebas Continuas[10]. Por lo que las dos nuevas prácticas cuentan con el mismo nivel de significancia que las tres anteriores. Dándoles gran importancia en el proceso de desarrollo de una aplicación, razón por la cual fueron elegidas para realizarles una guía de trabajo. Además se tuvo en cuenta que independientemente de su fusión no se le realizaron cambios a su funcionamiento o ubicación dentro del proceso de desarrollo. Lo que significa que su funcionamiento original ya era el óptimo para lograr el mejor resultado posible con su utilización. La Figura 4 ejemplifica cómo se fusionaron las tres prácticas originales para formar las dos actuales.

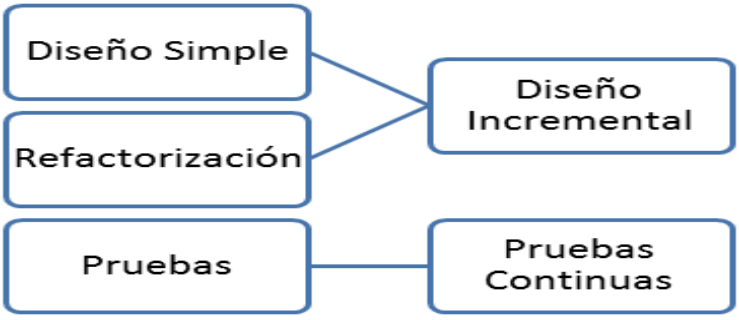

Figura 4. Fusión de prácticas. Fuente: Elaboración propia.

1.3 Las aplicaciones compuestas y los servicios.

En opinión de la autora una aplicación compuesta es aquella que se compone y nutre de varias aplicaciones informáticas. Diseñándose de manera tal que todas se encuentren acopladas de manera flexible y en constante comunicación entre sí, de modo que la aplicación compuesta pueda consumir información y servicios de las otras. Brindando por sus características una serie de beneficios tanto a los usuarios como a sus desarrolladores. En la Figura 5 se reúnen los tres principales beneficios que hacen de estas aplicaciones un elemento importante en el desarrollo de software actual.

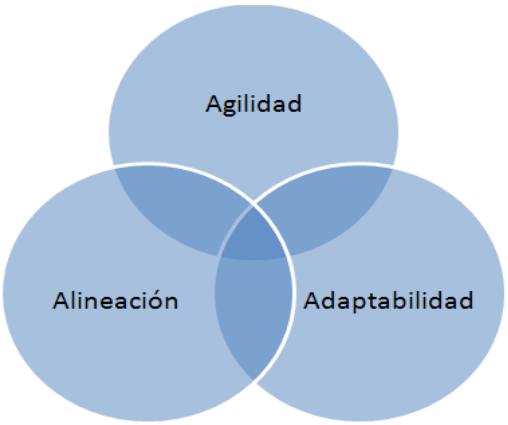

Figura 5. Beneficios del uso de aplicaciones compuestas. Fuente: Elaboración propia.

Independientemente de las características y beneficios de las aplicaciones compuestas, actualmente existen motivaciones para la realización de software orientado a servicios. Sucede que el mercado empresarial va en aumento de igual manera que el mercado informático; viendo reflejado el primero en el segundo una oportunidad para su propio éxito con los clientes y la competencia.

Una de estas motivaciones es la de crear software de forma ágil y que se adapte a las necesidades del usuario. Esto va impulsado por la visión de las empresas consumidoras que buscan soluciones integrales que se adapten a sus expectativas de crecimiento de una manera flexible.[11]. Para otras empresas, la prioridad es fortalecer a los usuarios del negocio para que sean ellos quienes reaccionen de un modo ágil ante los entornos de negocios de rápido cambio y que sobresalgan comercialmente [23].

Por otro lado esta tendencia que se da a escala mundial, requiere una evolución del mercado TIC[3] [11], evolución que según estudios que sean realizado, arrojan resultados donde según los cálculos predictivos basados en el crecimiento actual de este sector, para el año 2015 el porciento de la economía mundial que dependerá de la producción de software será el 41% [24], proporcionando casi la mitad de los ingresos monetarios a nivel mundial. La figura siguiente muestra una gráfica donde se ejemplifica este crecimiento.

[3] Mercado de la Tecnología Informática y las Comunicaciones.

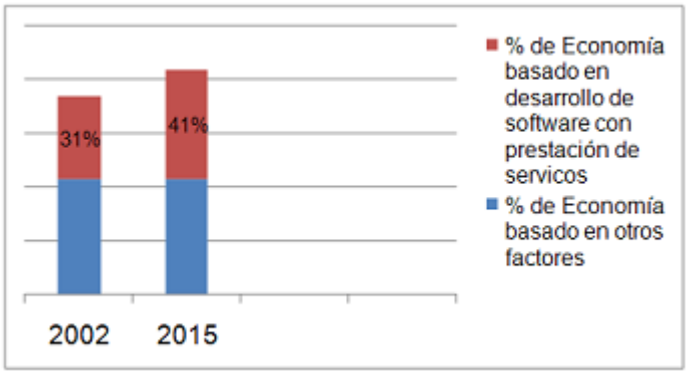

Figura 6. Economía mundial basada en el desarrollo de software con prestación de servicios. Fuente: Informe de vigilancia tecnológica “Tecnología de software orientadas a servicios”.

Después de analizar la información antes presentada se aprecia mejor la importancia de las aplicaciones compuestas en este nuevo enfoque se software orientado a servicios. Ya que estas con sus características y beneficios encajan a la perfección como la solución a las necesidades de las empresas. Con la alineación estas aplicaciones ofrecen la posibilidad de trasladar el debate de la reutilización desde el dominio técnico hacia el dominio de negocios, librando a las empresas de los confines cerrados de aplicaciones aisladas y sus desarrolladores, permitiéndoles definir comportamientos optimizados por sus negocios mediante metadatos y flujos de proceso. Mientras que la adaptabilidad y la agilidad permiten que las aplicaciones que se creen sean suficientemente flexibles para adecuarse a los cambios del mercado rápidamente.

Las aplicaciones compuestas cuentan con un proceso de desarrollo de cuatro etapas que puede distribuirse entre varios roles del equipo de desarrollo y de la administración de aplicaciones [25]. De manera general se trata del diseño, creación, ensamblaje y conexión y finalmente implantación de la aplicación. Sus componentes pueden ser un vínculo a una vista, un conjunto de marcos, un documento [26], formularios, campos, páginas, carpetas y guías [27] u otro tipo de componente. Estos deben ser reutilizables de manera que una vez diseñados puedan ser utilizados en cualquier aplicación compuesta donde sean necesarios. Por otro lado si la aplicación no tiene una interfaz gráfica su proceso de desarrollo

es más sencillo ya que solo es necesario obtener los contratos de los servicios (WSDL[4]), para a través de ellos utilizar las funcionalidades de los servicios.

Uno de los tipos de aplicaciones compuestas de mayor aceptación en la actualidad son los mashups. Que son aplicaciones web que acceden a datos o servicios de terceros y los combina para crear una nueva aplicación [28], como se aprecia en la Figura 7 sobre el funcionamiento de una aplicación mashups. Este tipo de aplicación compuesta ha alcanzado tal grado de aceptación entre los creadores de aplicaciones y consumidores, que en el año 2008 a tan solo cinco años desde de su aparición en el mercado, ya se creaban tres mashups diarios y existían 9 categorías distintas en el mercado: mapas, fotos, búsqueda, compras, deportes, viajes, videos, noticias y mensajería, predominando en la categoría de mapas donde dominaban el 44 % del mercado de aplicaciones de mapas del mundo [28]. Tendencia que ha ido en aumento hasta la actualidad.

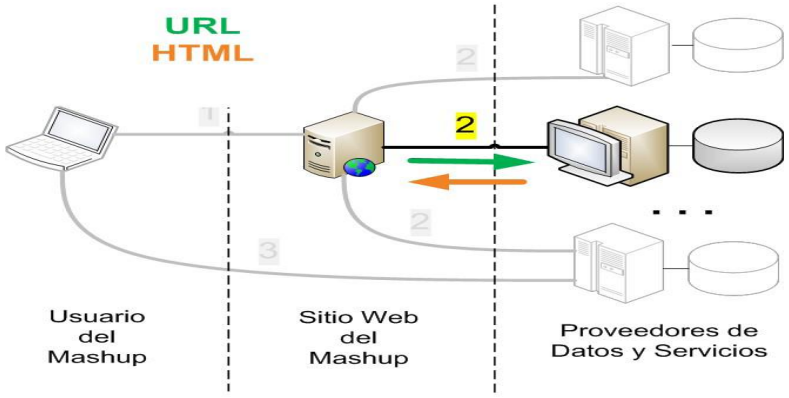

Figura 7. Flujo de datos y servicios de una aplicación mashups. Fuente: “Mashup, sumando en la red”.

1.4 Desarrollo de aplicaciones compuestas usando XP.

El desarrollo de una aplicación orientada tanto a consumir como a brindar servicios es un proceso complejo por varias razones y son estas razones las que llevan a ver en XP una excelente metodología para ser utilizada en el desarrollo de estas aplicaciones. Una de estas razones es la importancia de ofrecer resultados que sean medibles [11], otra es la necesidad de que la aplicación sea ágil y se adapte a las necesidades de usuarios y clientes [23].

[4] WSDL. Documento XML que describe ciertas características propias de un servicio web, así como su localización y métodos que soporta.

El tener resultados que sean medibles nos dirige a la ingeniería del software orientada al valor y no al control. Eso quiere decir que en un desarrollo de este tipo, la decisión de la implementación de un nuevo requisito tiene sentido si el valor añadido al producto lo merezca. Por lo tanto, en el ámbito del software orientado a servicios el concepto de ingeniería software orientado al valor que propone XP, donde tiene gran importancia crear un valor inmediato para presentar al cliente, cobra todo su sentido[11]. Mientras que el hecho de que las aplicaciones compuestas se construyan de manera ágil y dinámica, vuelve conveniente la utilización de una metodología de desarrollo ágil que permita realizar prototipos rápidamente para ser entregados al usuario y que el provea la retroalimentación necesaria para mejorar la aplicación de manera iterativa [12], otro ejemplo de la conveniencia de utilizar XP para desarrollar estas aplicaciones, ya que esta metodología plantea un proceso de desarrollo ágil y caracterizado por la realización de pequeñas entregas del producto al cliente en poco tiempo buscando la retroalimentación y la complacencia del cliente.

Por otro lado el centro Enterpriese Mashups de la compañía IBM, centro de producción de mashups mediante el uso exclusivo de la metodología XP. Considera que XP brinda múltiples y grandes beneficios a la realización de software. Sin embargo estos beneficios pueden ser aún mayores si se utiliza esta metodología en la creación de aplicaciones informáticas que se rijan por el paradigma de reusabilidad de código, como las aplicaciones compuestas. Según estudios realizados en dicho centro los mashups y la metodología XP coinciden en cuatro conceptos fundamentales: diseñar para entregar, reducción de la incertidumbre en el proyecto y del tiempo de duración del mismo, acceso a la información en el lugar donde se encuentre y reusabilidad de código[13].

Además de IBM otras empresas de gran influencia en el mercado informático como Nokia han decidido adoptar las metodologías ágiles para el desarrollo de sus aplicaciones, con resultados excelentes para los equipos de trabajo. A modo de ejemplo se muestran a continuación datos recopilados en una encuesta realizada en Nokia Networks dentro del proyecto Agile (proyecto donde se realizó la adopción de metodologías ágiles para el desarrollo de software) [11], del cual:

- El 10% de los encuestados dijo que volvería a los antiguos métodos.
- El 20% de los encuestados no ve diferencia en el cambio de metodología.

- Y un contundente 70% dijo que no volvería a trabajar como lo hacía antes.

Tras el análisis realizado en este trabajo, la autora del mismo ha llegado a la conclusión de que la metodología XP es la más indicada para afrontar el desarrollo de aplicaciones compuestas. Sin embargo a pesar de todos estos estudios que indican resultados positivos acerca del uso de XP para el desarrollo de aplicaciones compuestas, hasta el momento no se ha encontrado en la bibliografía consultada ni información relacionada explícitamente con la etapa de diseño de estas aplicaciones, ni información de XP sobre el desarrollo específicamente de aplicaciones compuestas. Es por esto que se plantea utilizar las prácticas de la metodología XP vinculadas al diseño de aplicaciones informáticas y realizarles variaciones a su flujo de trabajo; para crear una especificación de su funcionamiento que pueda ser aplicada en la etapa de diseño del proceso de creación de aplicaciones compuestas; como muestra la Figura 8. De manera que las prácticas Diseño Incremental y Pruebas Continuas adapten sus actividades, roles y artefactos en función del diseño de componentes y elementos necesarios para lograr mejorar el proceso de diseño de las aplicaciones compuestas, a través del proceso de desarrollo ágil que plantea la metodología XP.

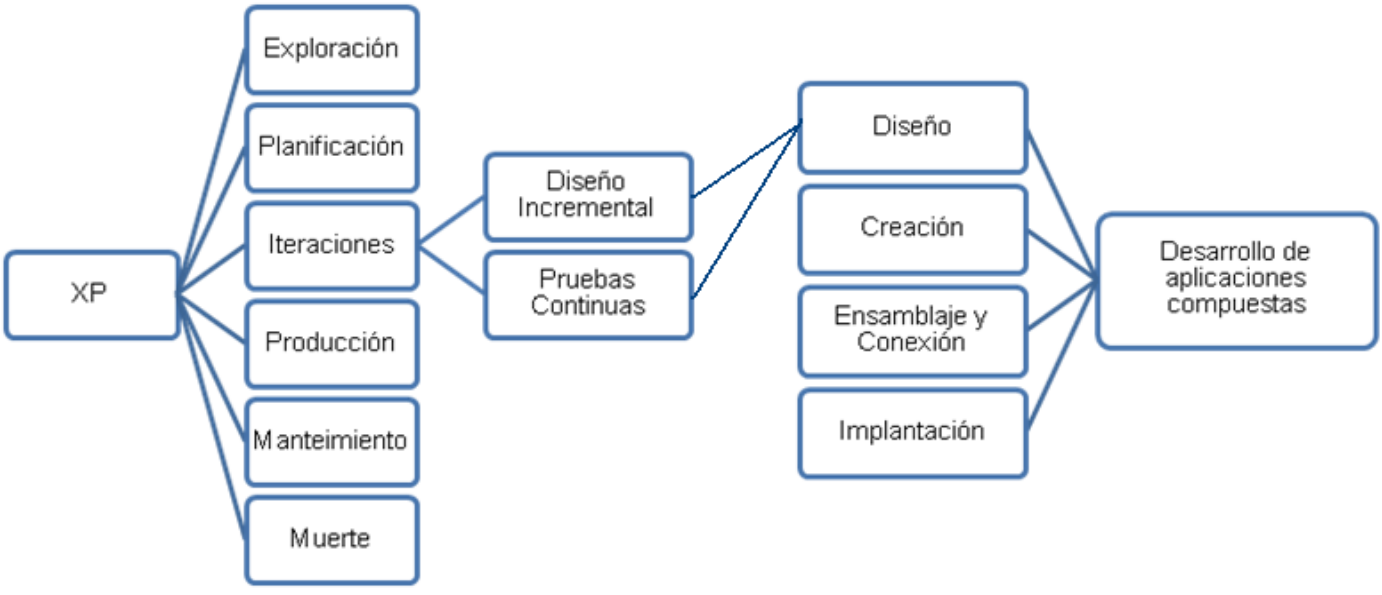

Figura 8. Vinculación de las prácticas Diseño Incremental y Pruebas Continuas con la etapa de diseño de aplicaciones compuestas. Fuente: Elaboración propia.

1.5 Conclusiones parciales.

En este capítulo se realizó un estudio del proceso de desarrollo de software de manera ágil, el cual fue ejemplificado a través del análisis de la metodología XP y su vinculación con el desarrollo de aplicaciones compuestas. Metodología la cual a través de su desarrollo basado en las relaciones interpersonales, la agilidad y la adaptación al cambio ha demostrado su efectividad sobre todo en proyectos con requisitos muy variables como es el caso de las aplicaciones compuestas. Debido a lo cual se concluyó que a la hora de utilizar una metodología para desarrollar aplicaciones compuestas, debería usarse XP, por las facilidades que brinda al proceso de desarrollo y las características que comparte con estas aplicaciones.

Capítulo 2: Propuesta de guía de trabajo de las prácticas Diseño Incremental y Pruebas Continuas de la metodología XP, para la etapa de diseño de aplicaciones compuestas

En este capítulo se realiza la propuesta de una guía de trabajo sobre las prácticas Diseño Incremental y Pruebas Continuas de la metodología XP. Para lo cual se le realizan modificaciones a su funcionamiento de manera que sus actividades, roles y artefactos queden adaptados al proceso de diseño de una aplicación compuesta.

2.1 Estructura de la Guía de trabajo.

A la hora de desarrollar una guía de trabajo es necesario tener claro cuáles son los elementos que la componen, cuál es la función de cada elemento dentro de la guía y cuál es su relación con el resto de los elementos que la forman. Por estas razones a continuación se muestra un esquema de la estructura que presentan los elementos que componen la guía que forma parte del resultado de esta investigación.

Figura 9. Estructura de la guía de trabajo. Fuente: Elaboración propia.

2.1.1 Elementos de la Guía de trabajo.

Los **roles** son los encargados de realizar las tareas o actividades, llenar y utilizar los artefactos de entrada y salida. En el caso de la guía que se propone en este trabajo investigativo teniendo en cuenta las características de la metodología XP y del desarrollo de aplicaciones compuestas se definieron tres roles, que fueron elegidos a través del estudio realizado, el cual arrojó que además de las competencias que debían tener los roles que utiliza normalmente la metodología XP para la etapa de diseño, era también necesaria la modificación de las actividades que debe realizar el rol de programador, otorgándosele tareas específicas del diseño de aplicaciones compuestas. Estos roles son:

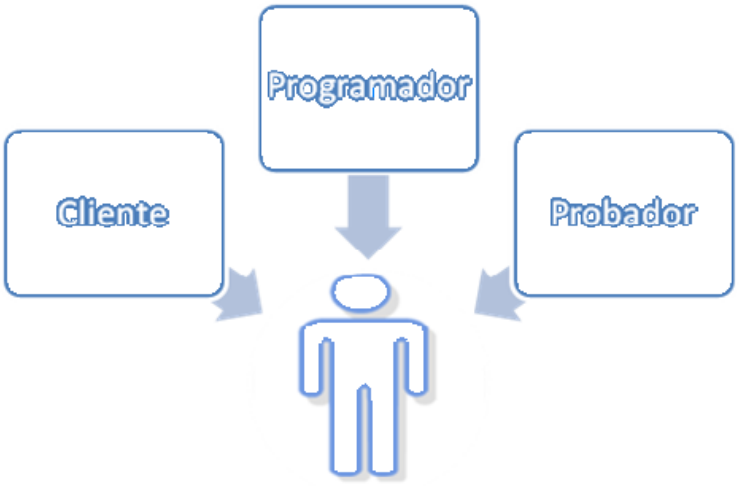

Figura 10. Roles de la guía de trabajo. Fuente: Elaboración propia.

- Programador: Escribe las pruebas unitarias, produce la lógica de la solución y realiza el diseño de la aplicación compuesta. Es importante que exista una buena comunicación y coordinación entre los programadores y otros miembros del equipo[17].
- Cliente: Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio[6].
- Probador: Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas [22].

El **artefacto** es una pieza de información producida, modificada y utilizada en un proceso. Es un producto tangible del proyecto que es utilizado por los roles como entrada para la realización de sus actividades y a la vez son el resultado de las actividades realizadas por los roles[29]. En esta guía se van a utilizar artefactos definidos por la metodología XP, y otros que serán incluidos por su necesidad para el proceso de diseño de aplicaciones compuestas.

- Diagrama de secuencia. Artefacto de salida en el cual se va a modelar la interacción entre las clases y objetos a lo largo de cada funcionalidad de la aplicación [12].
- Diagrama de actividades. Este es un artefacto de salida en el cual se van a mostrar los flujos de trabajo operacionales de la aplicación [12].
- Historia de Usuario. Es un artefacto de entrada, donde el cliente plasma los requerimientos básicos de su sistema en forma de una descripción de la funcionalidad.
- Archivo de pruebas. Artefacto de salida. Son los contenedores del código de pruebas de aceptación.
- Registro de datos de entrada. Artefacto de entrada. Es una lista con los datos que se le pueden pasar como entrada a las pruebas de aceptación y que es actualizado después con los datos que finalmente sean utilizados.
- Lista de tareas. Artefactos de salida. Es una lista que puede ser de papel o digital, donde se plasman las tareas en que han sido divididas las Historias de Usuario.
- Archivo del diseño. Artefactos de salida. Son los contenedores de la implementación que realicen los programadores del diseño.
- Planilla de especificación de servicios que consume. Artefactos de salida en el que se especifica cada servicio que consume la aplicación.
- Planilla de especificación de los componentes. Artefacto de salida en el que se especifica el diseño y propiedades de cada componente que conforma la aplicación[26].
- Guía a seguir para el proceso de refactorización. Artefacto de entrada. Es una lista de pasos a seguir que ayuda a realizar el proceso de refactorización de forma organizada [18].
- Planilla de pruebas de aceptación pasadas exitosamente. Artefacto de salida. Es una planilla de pruebas de aceptación que el sistema ha pasado exitosamente.
- Catálogo de Refactorización. Artefacto de entrada. Es un catálogo de refactorizaciones funcionales y óptimas, que los programadores pueden utilizar para mejorar el código de la aplicación[18].

- Planilla de pruebas de aceptación. Artefacto de salida. Es una planilla que contiene las pruebas que se ha decidido por mutuo acuerdo de programadores y clientes realizarle a la aplicación.

Una **actividad** es la unidad de trabajo que puede ejecutar un individuo en un rol específico. Tiene un propósito claro y se expresa en términos de actualizar artefactos. Su tiempo de realización es generalmente de horas o pocos días [29]. En el caso específico de esta guía se cuenta con un total de 18 actividades, divididas entre los pasos que se definen para la realización de ambas prácticas. Destinadas a dirigir de manera detallada el proceso de diseño de una aplicación compuesta.

2.2 Guía de trabajo con las prácticas Diseño Incremental y Pruebas Continuas, para el diseño de aplicaciones compuestas.

Cuando la aplicación llega a la fase de Iteraciones ya ha pasado por un proceso de estimación para determinar cuánto tiempo debe tardar su desarrollo. Además el cliente ha determinado el nivel de prioridad que él considera que tiene cada Historia de Usuario. Tras estas acciones la aplicación se encuentra lista para entrar en un proceso de diseño de sus funcionalidades, teniendo en cuenta los intereses y necesidades del cliente. Proceso que se llevará a cabo a través de las prácticas Diseño Incremental y Pruebas Continuas [17].

En la práctica Diseño Incremental se deberá idear la solución más simple para que la aplicación pueda funcionar. La complejidad innecesaria y el código extra deberán ser removidos inmediatamente, además de mejorar su legibilidad y hacerlo más flexible para facilitar los posteriores cambios[12]. Sin embargo XP plantea un modelo de desarrollo guiado por las pruebas, que permite corregir los errores inmediatamente cuando ocurren y no en etapas posteriores. Específicamente en el caso de las aplicaciones compuestas la no realización de las pruebas según el modelo de XP puede ocasionar problemas en el consumo de servicios, lo que influye directamente en la respuesta que se le vaya a dar al cliente, es por esto que la práctica Pruebas Continuas se ejecuta conjuntamente a la práctica Diseño Incremental.

La práctica Pruebas Continuas para lograr un mejor rendimiento de su funcionalidad tiene varias aristas de funcionamiento, las cuales son: Los clientes ayudan a escribir las pruebas de aceptación antes que se escriba el código. De igual manera los programadores escriben pruebas para el código, todavía no escrito,

llamadas pruebas unitarias. El propósito de ambos tipos de prueba es aligerar el proceso de codificación, ya que al hacer las pruebas el objetivo real de los programadores no es cumplir un requerimiento, sino pasar dichas pruebas con éxito. Es importante mencionar que ambos tipos de pruebas pueden ejecutarse tanto de manera automática para evitar errores humanos[12], como manual.

2.2.1 Tipos de pruebas.

Con el objetivo de esclarecer la función de las pruebas en el proceso de desarrollo, en este trabajo se explicará las características de ambos tipos de pruebas, pero en la guía de trabajo solo se tratará el funcionamiento de las pruebas de aceptación ya que solo estas se ejecutan durante el diseño de la aplicación.

Pruebas de aceptación: Estas pruebas son hechas por el Cliente y el Probador. Su objetivo es verificar todo el sistema, o una gran parte de él. Este tipo de pruebas, comprueba la calidad de aspectos como: que la aplicación funcione correctamente, que cargue correctamente, la actuación del sistema, su compatibilidad, que no haya problemas con su instalación, que no existan agujeros en su seguridad, entre otros. Se le pueden aplicar al sistema lo mismo una vez que todo el diseño esté terminado, que durante el proceso de diseño, cada vez que se termine de diseñar una funcionalidad[21].

Pruebas de unidad: Estas pruebas son hechas en el mismo período de tiempo que las pruebas de aceptación, solo que sus creadores son los programadores. Este tipo de pruebas a diferencia de las anteriores, están dedicadas a comprobar el código de la aplicación. Cada una de las pruebas unitarias que se le realicen a un sistema deberá verificar una sola clase, o un pequeño conjunto de clases. Dichas pruebas se deberán realizar principalmente en la fase de Producción, para garantizar que el código del sistema funcione correctamente[21].

2.2.2 Realizar el diseño de la aplicación.

Es importante aclarar que en momentos anteriores a la realización del diseño, el equipo de trabajo debe reunirse para determinar y crear una Estrategia de Diseño a seguir. Estrategia que regirá el proceso de diseño mediante la creación de un conjunto de previsiones sobre fines y procedimientos que forman una secuencia lógica de pasos o fases a ser ejecutadas, para alcanzar los objetivos planteados.

Con la Estrategia de diseño creada, la aplicación está lista para ser diseñada, solo queda aplicar dicha estrategia y empezar con el diseño de la aplicación. Lo primero es tomar la Historia de Usuario que el cliente determinó que tenía la mayor prioridad, para empezar su proceso de diseño mediante los pasos siguientes.

Paso 1: Análisis de la lógica de la solución.

En este paso inicial será el rol de programador el que tendrá el protagonismo. Las actividades que se desarrollarán en este paso, estarán dirigidas a que los programadores, entiendan y se familiaricen con la lógica de ejecución del sistema [7].

Roles:

- Programador.

Artefactos:

- Historia de Usuario (HU).
- Diagrama de secuencia.
- Diagrama de actividades[12].

Actividades:

- **Análisis de las Historias de Usuario:** En este paso se realiza un análisis previo de cuáles serán las funcionalidades básicas que se deben cumplir con el desarrollo de las HU. En este momento inicial se analizan cuáles serán los posibles objetos que se utilizarán en el diseño, los métodos que se deberán realizar. También se realiza un análisis de los servicios que formarán parte de la aplicación, si se brindarán, consumirán o ambas cosas, analizando además qué componentes serán necesarios para la realización de la aplicación, ejemplos de estos componentes son: formularios, campos, páginas, carpetas y guías [27]. Para lograr un mejor entendimiento del plan de ejecución futuro se realizan diagramas de secuencia y actividades.

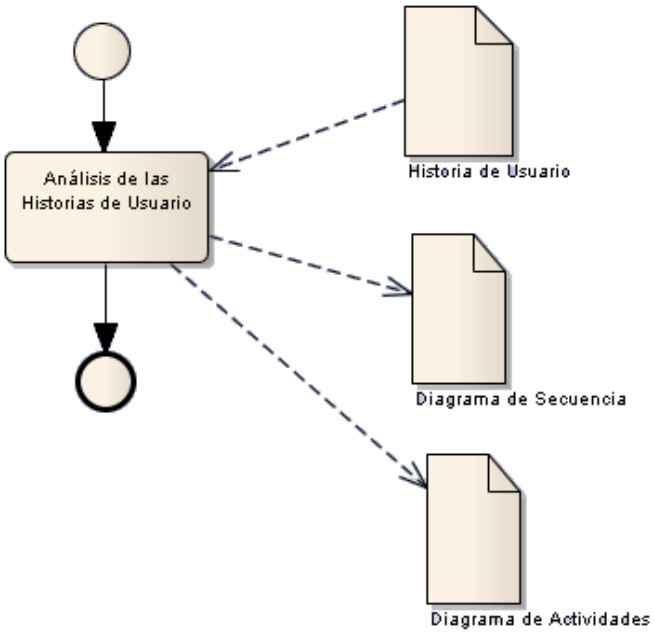

Figura 11. Análisis de la lógica de la solución. Fuente: Elaboración propia.

Paso 2: Crear las pruebas.

Una vez determinadas las funcionalidades principales que se deben lograr con la HU, se entra en el segundo paso de la guía, donde la responsabilidad de realización de las actividades se comparte entre los roles Programador, Probador y Cliente. Este es un paso importante, puesto que en él se van a escribir las pruebas de aceptación con las que se va a determinar si el diseño cuenta con la calidad requerida [30].

Roles:

- Probador.
- Cliente.
- Programador.

Artefactos:

En este paso en particular, los artefactos que se utilicen dependerán por completo del tipo de prueba que se vaya a realizar, por lo que solo se mencionarán los más comunes, teniendo como base que son los que se manejan en las pruebas más utilizadas[21].

- Historias de Usuario.

- Archivo de pruebas de aceptación.
- Registro de datos de entrada.
- Lista de pruebas de aceptación.

Actividades:

- **Análisis del sistema con el cliente:** El miembro del equipo que se encuentra desempeñando el rol de Probador se reunirá con el Cliente para discutir acerca de las funcionalidades y características que el cliente desea que tenga el sistema.

- **Análisis de las posibles pruebas de aceptación:** Una vez que ya ambos tengan toda esta información clara, el probador deberá presentarle al cliente varias opciones de pruebas de aceptación, en caso de que el cliente no conozca ninguna, ni haya pensado en alguna variante por él mismo.

- **Análisis de las herramientas para las pruebas de aceptación:** Cuando ya se hayan determinado la o las pruebas de aceptación que se vayan a realizar, el probador deberá explicarle y enseñarle al cliente, el funcionamiento de la o las herramientas con las que se va a trabajar para la creación de la prueba, en caso de que se necesite alguna herramienta.

- **Creación de las pruebas de aceptación:** Con todas las bases para el proceso de realización listas, el cliente y el probador pueden empezar a escribir las pruebas. El cliente puede escribir las pruebas él mismo, con la asistencia del probador para los temas que él desconozca. O el probador puede escribir las pruebas, guiándose por los criterios del cliente.

- **Divulgación de las posibles pruebas de aceptación a los programadores:** Ya con las pruebas de aceptación terminadas, el probador tiene la tarea de informarles a los programadores que estarán implicados en el desarrollo del sistema cuáles son las pruebas de aceptación que se crearon para probar el sistema.

- **Análisis de las pruebas de aceptación por parte de los programadores:** Los programadores tienen el derecho a negarse a la realización de alguna prueba que ellos consideren innecesaria o

inconveniente. Debido a esto, cuando el probador le presenta las pruebas, ellos deben analizarlas, y determinar si están de acuerdo o no con su ejecución.

- **Acuerdo de las pruebas de aceptación:** En caso de que los programadores no estén de acuerdo con alguna prueba, es deber del probador lograr que exista un acuerdo mutuo entre programadores y clientes en cuanto a las pruebas que se le van a realizar a la aplicación, de manera tal que las dos partes estén conformes [30].

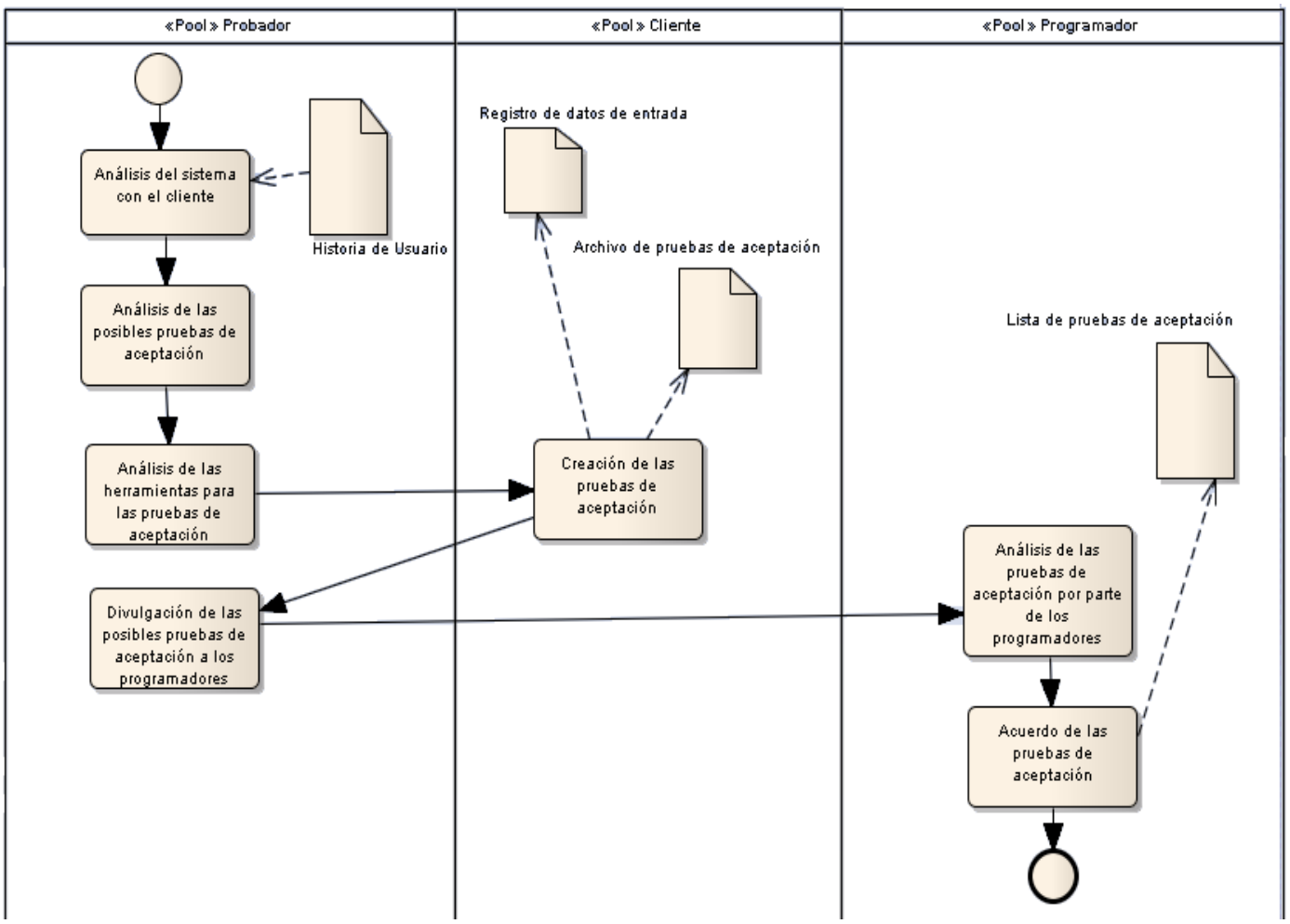

Figura 12. Creación de las pruebas de aceptación. Fuente: Elaboración propia.

Paso 3: Diseñar.

Después de haber escrito las pruebas, se empieza el proceso de diseño de la aplicación, el cual se realiza por la vía Prueba_Primero-Codifica. Este proceso, se realiza de manera gradual e incremental, manteniéndose abierto a cualquier posible cambio. En este paso es donde el desarrollo de la aplicación pasa por más modificaciones hasta crear el diseño deseado. Los programadores tienen una gran participación en este paso, ya que estará realizando la mayor parte de las actividades.

Roles:

- Programador.
- Desarrollador de Componentes.
- Probador.

Artefactos:

- Historias de Usuario.
- Lista de tareas.
- Archivo del diseño.
- Plantilla de especificación de servicios que se consumen.
- Plantilla de especificación de los componentes.
- WSDL de servicio.

Actividades:

- **División de las Historias de Usuario en tareas:** Los programadores dividen la HU en tareas, dependiendo del número de funcionalidades que se deban diseñar, de manera que las tareas sean lo más atómicas posibles [17], para poder diseñarlas. El proceso de diseño de cada tarea debe realizarse siguiendo las pautas trazadas en la Estrategia de Diseño, por esta razón el diseño e implementación del diseño que se realicen debe estar enfocados únicamente en tener las condiciones para pasar exitosamente las pruebas destinadas a probar su calidad, realizadas en el paso anterior [7].

- **Diseño de la aplicación:** Esta actividad está destinada a crear el diseño general de la aplicación. Aquí se especifican los servicios a los que va estar vinculada la aplicación y las relaciones entre ellos mediante propiedades de conexión y elementos de diseño, además de las propiedades de

salida y “acciones” a realizar. El término “acción” se emplea para designar la conexión de un componente que consume el valor de una propiedad [26].

- **Diseño de componentes:** Esta actividad se basa en diseñar los componentes que se habían escogido anteriormente para conformar la aplicación, siempre recordando que el diseño que se realice de los componentes debe tener la propiedad de reusabilidad. Para diseñarlos deben realizarse las siguientes acciones:
    - Determinar las propiedades que publicará el componente.
    - Determinar las acciones que el componente realizará cuando se conecte con una propiedad de otro componente.

- **Diseño de servicio:** En el caso de que se necesite consumir un servicio que no esté implementado aun en ningún repositorio de servicios. El programador deberá analizar el funcionamiento que debe ejecutar dicho servicio e implementarlo.

- **Diseño de las tareas por el probador:** Esta actividad es opcional y su realización depende de la decisión del equipo de trabajo. Se basa en que el probador ayude al programador a escribir el código del diseño, ya que él tiene conocimiento de las pruebas que se le realizarán al mismo [28], incrementando las posibilidades de que la implementación que se haga del diseño, tenga la calidad requerida para pasar exitosamente las pruebas de aceptación.

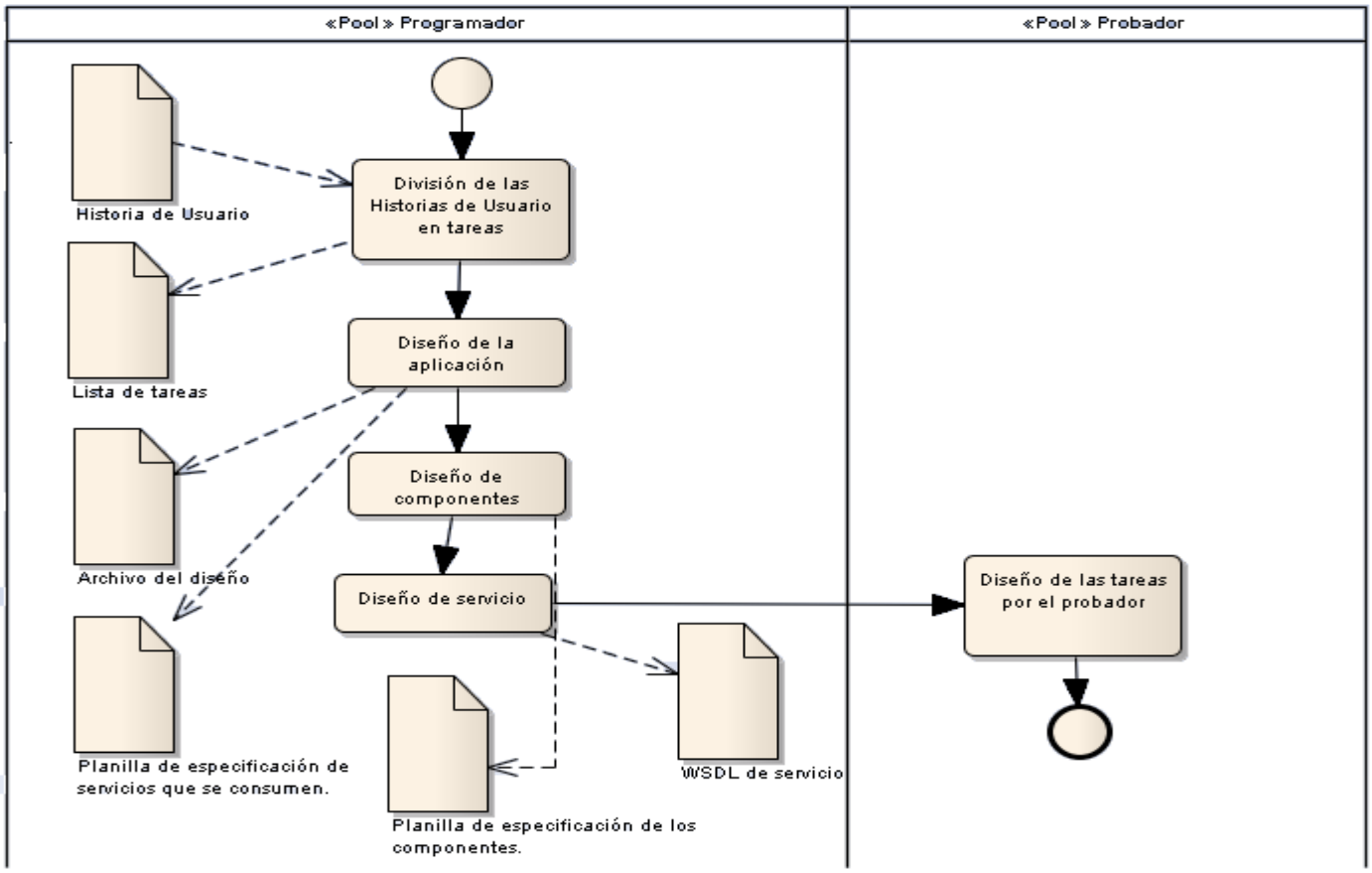

Figura 13. Diseño de la aplicación. Fuente: Elaboración propia.

Paso 4: Modificar y mejorar

Las pruebas de aceptación pueden ser ejecutadas en la culminación de cada tarea de manera opcional, pero una vez que se haya terminado el diseño de las funcionalidades es de carácter obligatorio ejecutarlas todas, ya que aún se debe comprobar que no existen errores en el funcionamiento conjunto; además de verificar si es necesario hacerle alguna modificación o mejora al diseño y la implementación de la funcionalidad, en aras de volverlo más simple. Este paso se conocía en la primera versión de XP como Refactorización.

Roles:

- Programador.
- Cliente.

Artefactos:

- Guía a seguir para el proceso de refactorización.
- Lista de pruebas de aceptación pasadas exitosamente.
- Catálogo de Refactorización.
- Registro de datos de entrada.

Actividades:

- **Aseguramiento del Catálogo de Refactorización:** Lo primero que se debe hacer es asegurarse de tener un Catálogo de Refactorización para aplicarlo en el momento en que sea necesario. Y una guía a seguir durante el proceso.

- **Comprobación de interoperabilidad entre los componentes:** Es necesario comprobar que los componentes que conforman la aplicación, se comunican de la manera correcta. Y reaccionan como es debido a la acción de cada uno con respecto a otro. Verificar que su tiempo de respuesta es aceptable con respecto a las necesidades de la aplicación.

- **Comprobar el funcionamiento del servicio:** Es necesario comprobar que el servicio funcione de forma correcta. Ya que puede existir algún error interno, que ocasione que los resultados del mismo no sean los esperados. Un aspecto importante es que si los servicios no están disponibles y funcionales no se podrá llevar a cabo la composición.

- **Análisis del código:** Después deben analizar el código detenidamente, buscando cualquier detalle que pueda ser mejorado.

- **Mejoramiento del código:** Una vez encontrada alguna parte del código que pueda ser mejorada, se procede a utilizar el Catálogo de Refactorización. Se verifica si en el catálogo existe algún código que pueda sustituir al existente. De ser así se realiza el cambio. En caso de no encontrarse ninguno, los programadores deben realizar una refactorización propia.

Con el objetivo de ofrecer mayor claridad en este paso de ejecución de las pruebas de aceptación y mejoramiento del código del diseño, se presentarán algunas de las pruebas que se le realizan normalmente a las aplicaciones y a los servicios, aunque es importante aclarar que en ambos casos existen más pruebas para validar su calidad en cuanto a seguridad, rendimiento o integración, pero por su nivel de profundidad esas pruebas pertenecen a otras etapas del desarrollo de la aplicación y los servicios.

- **Pruebas de caja negra:** Estas pruebas se centran en los requisitos funcionales y pueden realizársele tanto la aplicación en general como a los servicios. Utilizando condiciones de entrada que ejerciten los requisitos funcionales. Es en estas pruebas donde se utiliza el artefacto Registro de datos de entrada.

- **Pruebas de diseño**. Estas pruebas se le realizan a los servicios, se realizan en su mayoría de manera manual y tienen el objetivo de verificar que el servicio cumpla con la interoperabilidad requerida.

- **Pruebas de seguridad:** Son pruebas destinadas a determinar los niveles de permiso de los usuarios, las operaciones de acceso a la aplicación o el servicio y el acceso a datos.

- **Pruebas de estructura:** Estas pruebas están enfocadas en asegurar que todos las conexiones funcionan correctamente y el contenido deseado es mostrado.

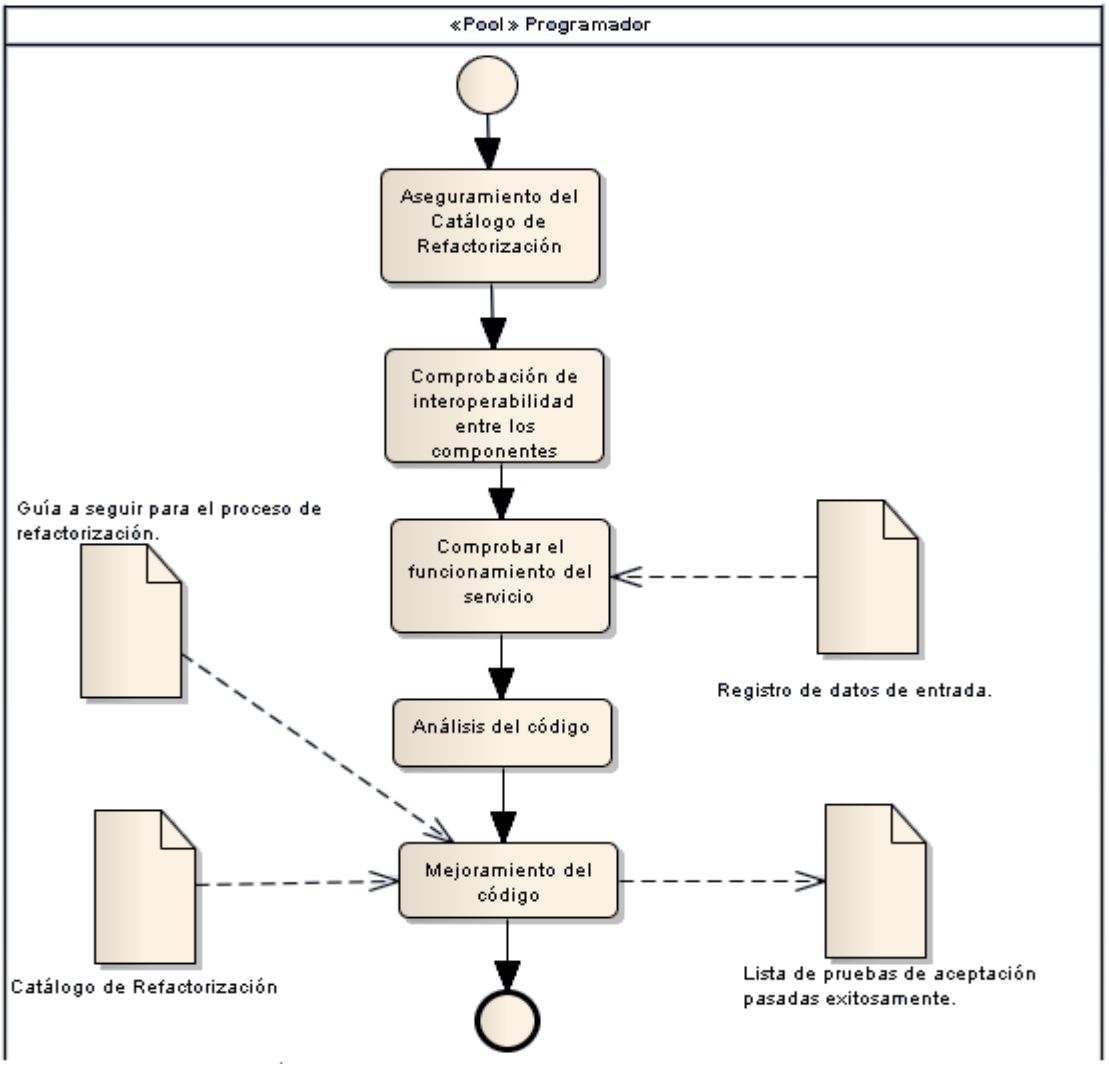

Figura 14. Refactorización del diseño. Fuente: Elaboración propia.

Este proceso de análisis del código y comprobación del código continua hasta que el sistema sea capaz de pasar con éxito todas las pruebas de aceptación, momento en el que se puede decir que el proceso de diseño se da por terminado. O sea que al terminar la ejecución de las prácticas Diseño Incremental y Pruebas Continuas, el sistema se encuentra en una condición en la que el cliente está complacido con el producto y se ha comprobado que funciona correctamente, por lo que está listo para entrar al proceso de construcción.

2.3 Conclusiones parciales.

En este capítulo se ha propuesto una guía de trabajo sobre las prácticas Diseño Incremental y Pruebas Continuas de la metodología XP. Con el objetivo de lograr agilizar y organizar el proceso de desarrollo aplicaciones compuestas mediante la filosofía de desarrollo que plantea esta metodología. Para esto se han definido los roles que deben participar en el funcionamiento de dichas prácticas, junto con las actividades que se deben realizar y los artefactos que deben utilizarse y generarse durante su ejecución.

Capítulo3: Validación de la guía

En el presente capítulo se llevar a cabo la validación de la guía de trabajo mediante la aplicación del método matemático Delphi y la realización de casos de estudio, con el objetivo de determinar la calidad, efectividad y funcionalidad de la de guía de trabajo propuesta.

3.1 Validación por la Técnica de Evaluación de Expertos.

La primera parte de la validación esta enfocada en determinar la calidad de la propuesta para lo cual se utilizó la Técnica Evaluación de Expertos, esta técnica está compuesta por métodos que se basan en la consulta a expertos. Los expertos son personas a las que se les ha evaluado previamente, y han demostrado tener un nivel de conocimiento elevado sobre el tema que van a validar[31]. Y de estos métodos se escogió en particular el método Delphi.

3.1.1 Método Delphi.

El Método de Delphi es una técnica de investigación que se utiliza con el objetivo de obtener una opinión confiable sobre un tema determinado, a través de las opiniones de expertos. Los expertos pueden ser un grupo de personas u organizaciones que tengan la capacidad de brindar opiniones, valoraciones, y conclusiones de un problema o tema específico. Además de hacer recomendaciones al respecto con un máximo de competencia[10]. Este método fue creado por Olaf Helmer y Theodore J. Gordon, durante la década de 1950. El objetivo del mismo era realizar predicciones con fines militares, pero a partir de la década de 1960 empezó a ser empleado en marcos de trabajos académicos y empresariales como técnica de previsión y consenso en situaciones de incertidumbre, donde no es posible utilizar otras técnicas apoyadas en información objetiva [30]. Actualmente es un método basado en identificar y/o priorizar preferencias o soluciones a problemas prácticos por parte de un grupo de expertos[31].

3.1.1.1 Funcionamiento Del Método Delphi.

La ejecución del método Delphi se basa en la realización de interrogantes a un grupo de expertos con la ayuda de cuestionarios sucesivos, a fin de poner de manifiesto convergencias de opiniones y deducir

eventuales consensos [30]. Para lograr el objetivo deseado es necesario seguir una línea de acción o el resultado podría ser erróneo. Esta línea está conformada por cuatro etapas fundamentales:

Formulación del problema: En un método de expertos, tiene gran importancia definir con precisión el campo de investigación, ya que si este es muy grande pueden haber imprecisiones en el nivel de conocimiento de este campo por parte de los expertos reclutados[30].

Elección de expertos: Cada experto debe ser elegido por su capacidad de encarar el futuro y por poseer conocimientos sobre el tema que le será consultado [30]. Para la elección de los expertos en este trabajo se analizaron los siguientes requisitos

- Graduado del Nivel Superior.
- Conocimientos sobre metodologías de desarrollo de software.
- Conocimientos sobre Aplicaciones Compuestas.
- Conocimientos sobre la metodología XP.
- Experiencia laboral.
- Capacidad de análisis y pensamiento lógico.

Como el método Delphi no plantea que se requiera un número exacto de expertos, sino que se puede contar con un rango entre 7 y 30 [32]. Para la validación de esta guía de trabajo se seleccionaron 19 posibles expertos. A los cuales se les realizó una primera encuesta con el objetivo de obtener información de su desempeño laboral y su nivel de conocimientos sobre el tema a validar. Dicho cuestionario se encuentra en el **Anexo 6**.

La selección de los expertos se hace de acuerdo a la valoración de sus competencias, sin embargo para esto es necesario **calcular el coeficiente de competencia (K)**, coeficiente que se calcula mediante la opinión del experto sobre su nivel de conocimiento acerca del problema que se está resolviendo y con las fuentes que le permiten argumentar sus criterios. Utilizándose la siguiente fórmula: **K = 1/2 (Ka + Kc)**. Donde **Kc** es el coeficiente de conocimiento o información que tiene el experto acerca del problema, valorado por el propio experto en una escala del 0 al 10 y multiplicado por 0.1. Mientras que **Ka** es el coeficiente de argumentación o fundamentación de los criterios del experto, obtenido como resultado de la

suma de los puntos alcanzados a partir de una tabla patrón, ver **Anexo** . Por último estos valores son insertados en tres intervalos predeterminados, para determinar el nivel del coeficiente de competencia de cada posible experto. En el caso particular de este trabajo se escogieron los expertos cuyo coeficiente de competencia estuvo en los dos primeros intervalos:

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.

Dicha selección de expertos finalmente proporcionó la siguiente distribución de porcentajes entre los niveles de conocimiento de los expertos. Donde de los siete expertos que finalmente se eligieron, 2 fueron calificados de nivel alto y 5 de nivel medio.

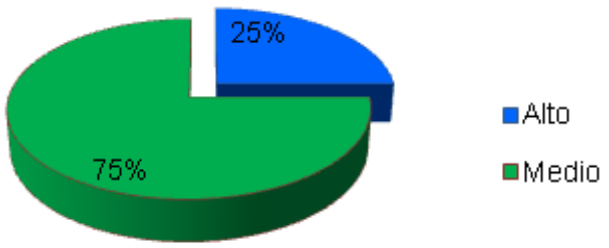

Figura 15. Porcentaje de nivel de competencia de los expertos seleccionados. Fuente: Elaboración propia.

Elaboración y lanzamiento de los cuestionarios, para validación de la propuesta: La elaboración de los cuestionarios debe realizarse de manera paralela a la selección de los expertos. Elaborándose de manera que faciliten la respuesta por parte de los consultados. Preferentemente las respuestas habrán de poder ser cuantificadas y ponderadas. Para después llevarlas a términos porcentuales y ubicar a la mayoría de los consultados en una categoría[32].

El cuestionario para validar la guía de trabajo aquí propuesta, consta de 9 preguntas, las cuales tienen como objetivos que los expertos den su opinión sobre la estructura, elementos y calidad de la guía de trabajo. Objetivos que se dividieron de la siguiente forma:

- En la pregunta uno se verifica la necesidad de la guía de trabajo propuesta.
- En las preguntas dos y tres se verifica la adecuación de las prácticas y la metodología escogidas.
- Mientras que las pregunta desde la cuatro hasta siete están destinadas a verificar la calidad de la propuesta en cuanto a: lógica y claridad de los pasos propuestos, correspondencia de las actividades y los roles con el proceso de desarrollo de aplicaciones compuestas y su necesidad en el mismo, correspondencia entre los artefactos y las actividades donde son generados.

Desarrollo práctico y explotación de los resultados: Debido a que la utilización del método Delphi en su formulación más teórica, supone tener que realizar tres o más encuestas para llegar a un consenso[33]. Para la validación de la propuesta aquí planteada se utilizará una versión del mismo. Donde el segundo cuestionario cuenta con 7 preguntas, cada una con 2 posibles respuestas (Si\No), a las que se les da una puntuación de 2 y 1 respectivamente. Donde 2 significa que la respuesta se ajusta al objetivo de la propuesta, mientras que 1 indica que existen problemas en la propuesta en cuanto al criterio evaluado[34].

Para determinar la concordancia entre el criterio de los expertos y poder arribar a un consenso en un menor número de rondas, se utilizó el coeficiente de Kendall (W) [34]. Este coeficiente mide el grado de asociación entre varios conjuntos de N entidades. Su valor oscila entre 0 y 1, donde 1 significa una concordancia de juicios total, y el valor 0 un desacuerdo total [35]. Finalmente tras realizar los cálculos pertinentes se concluyó que:

- En las preguntas 1, 3, 4, 5, 6 y 7 el 100% de los expertos estuvo de acuerdo con responder Sí.
- Presentándose discordancias solo en la pregunta 2, donde el 90 % de los expertos respondió que Sí, mientras que el 10% restante respondió que No.

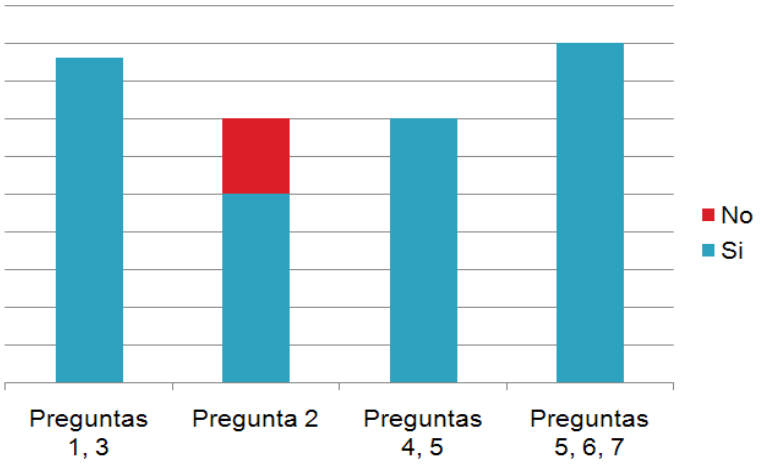

Figura 16. Grafico de aceptación de la guía de trabajo. Fuente: Elaboración propia.

Por lo que se puede concluir que la propuesta es válida, ya que los expertos estuvieron de acuerdo en responder que Sí el 90% de los casos.

3.2 Validación por caso de estudio.

La validación de la efectividad y la funcionalidad de la guía de trabajo se realizarán mediante la ejecución de dos casos de estudio. El primero se trata de una aplicación que brinda ofertas recreativas en función del estado del tiempo. Y el segundo es una funcionalidad de una plataforma de gestión de información que gestiona información sobre estudiantes.

3.2.1 Descripción del caso de estudio Cartelera Cultural.

La Universidad de las Ciencias Informáticas oferta atractivas opciones culturales de música, teatro y literatura de las principales instituciones culturales en Ciudad de La Habana y dentro de la UCI para el disfrute de la comunidad universitaria. Las mismas son ofrecidas por varios sitios entre los que se encuentran: los sitios de las facultades, la intranet, etc. de forma dispersa y que en ocasiones al usuario que desee conocer estas ofertas les resultaría tedioso la búsqueda de las mismas.

Si estuviese disponible una plataforma que permitiese el desarrollo personalizable de aplicaciones híbridas donde el usuario pueda obtener esa información y en dependencia del estado del tiempo

imperante en ese instante le ayudase a decidir si es factible la salida, en caso de que la salida pueda efectuarse le envíe una guía de las principales rutas de ómnibus para el traslado en la ciudad, así como compartir esa oferta con otra persona vía correo electrónico. Por otra parte, si las condiciones climatológicas lo impidiesen le ofertara otra opción, de manera que los usuarios estarían informados de lo que acontece y lograrían planificar actividades dentro de la brigada o de estimulación por parte de las organizaciones políticas y estudiantiles a sus militantes.

Para ello se realizará una aplicación híbrida que permita enviar por correo a un usuario las ofertas culturales y las rutas de los ómnibus metropolitanos en dependencia del estado del tiempo, así como la muestra de información de servicios que expone la UCI como: el menú del comedor para el día actual, así como las entradas del blog de SOA, que es uno de los blogs de nueva creación en la UCI y que aborda temas novedosos en relación con las arquitecturas orientadas a servicios muy utilizadas en el mundo de la informática[36].

3.2.1.1 Aplicación de la guía para el caso de estudio Cartelera Cultural.

Tras la aplicación de la guía de trabajo al caso de estudio Cartelera Cultural se concluyó que el proceso de diseño de la aplicación compuesta mediante los pasos determinados en la guía tuvo un tiempo de realización más corto (5 días) que mediante la utilización de la vía utilizada anteriormente (Intuitiva) para su realización (9 días), además de contribuir a la creación de un diseño mejor estructurado, la obtención de un menor número de inconformidades y un menor número de pruebas de aceptación con errores. Ahora se explicarán los resultados obtenidos en cada paso y se mostrarán algunos artefactos generados a modo de ejemplo.

Paso 1: En la realización del primer paso de la guía para el diseño de la aplicación que está destinado a conocer la lógica de la funcionalidad, se determinó que se iban a consumir tres servicios y se iban a brindar dos:

Consume: El estado del tiempo. Esta funcionalidad se basa en consumir el estado del tiempo que aparece en el sitio Octavitos.uci.cu.

Consume: El menú del comedor. Esta funcionalidad se basa en consumir el menú del comedor de cada día que aparece en el sitio Dragones.uci.cu.

Consume: El número de entradas al blog de SOA. Esta funcionalidad se basa en consumir las entradas que se registran en el Blog de SOA.

Brinda: Listado de películas. Esta funcionalidad se basa brindar un listado con las películas que se encuentran en el sitio Inter-nos.uci.cu, en dependencia del género de película que el cliente escoja en la aplicación.

Brinda: Ofertas recreativas. Esta funcionalidad propone un grupo de ofertas recreativas en caso de que el estado del tiempo permita salir de la casa.

Se determinó la utilización de los siguientes **componentes**: Lista desplegable, Test tarea, Botón, Formulario, Documentos Word y PDF, Tabla, Div.

Y se generaron un diagrama de actividades y el siguiente **diagrama de secuencia:**

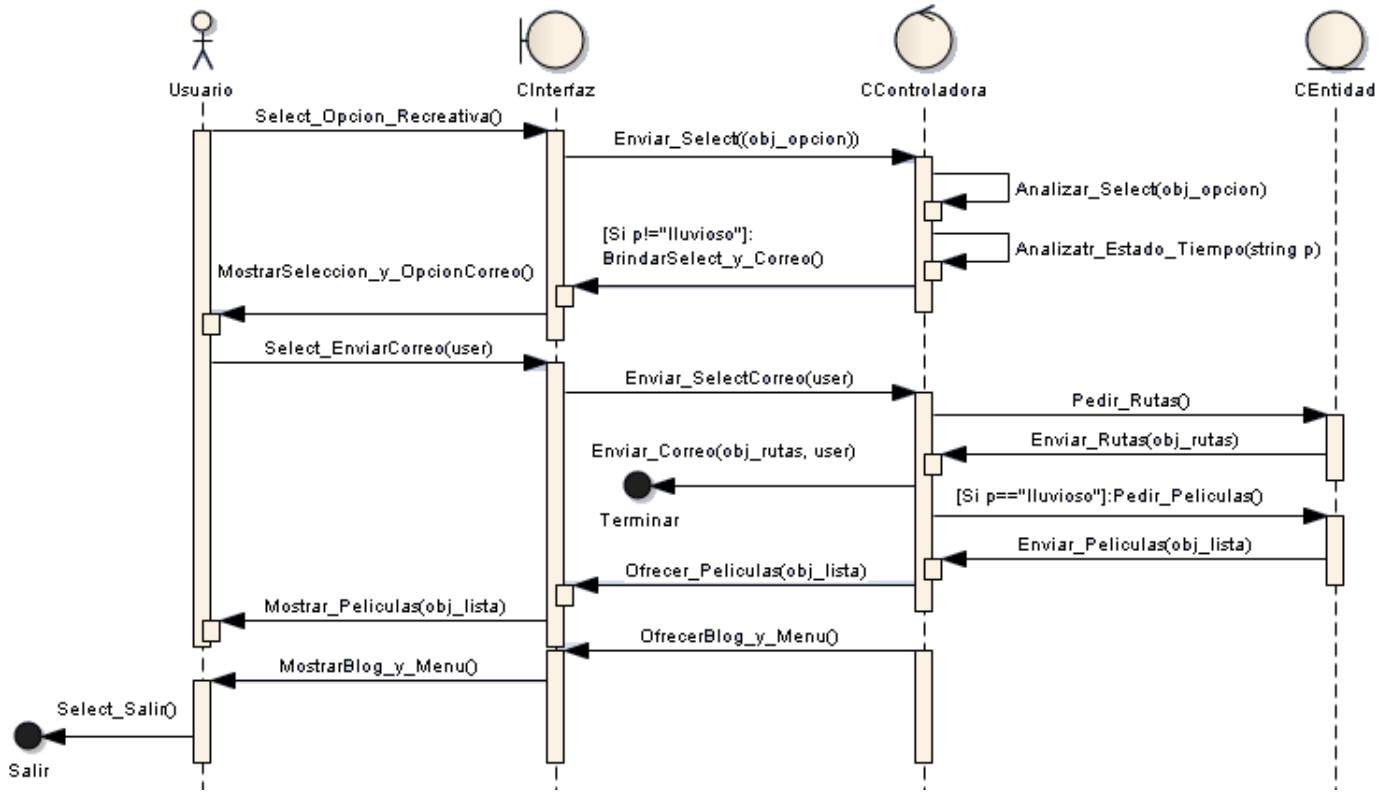

Figura 17. Diagrama de secuencia del caso de estudio Cartelera Cultural. Fuente: Elaboración propia.

Paso 2: En la utilización del segundo paso de la guía que está destinado a crear las pruebas para validar el funcionamiento de la aplicación. Se crearon las pruebas para comprobar que el usuario no dejara campos vacíos, que el usuario de correo que el usuario introduce sea válido, que los servicios consuman y muestren las informaciones que deben y que su tiempo de carga no sea excesivo (más de 5 segundos). No se generó ningún archivo de pruebas de aceptación, debido a la sencillez de las pruebas.

Tabla 1. Planilla de Prueba de aceptación. Fuente: Elaboración propia.

# Prueba: 1	**HU # 3**
Prueba: Validación de datos.	
Rol responsable: Programador: Dimelza Remedios Cruz.	
Descripción: Se le introduce una variable de valor nulo a la aplicación, y se espera que muestre un mensaje de error por introducir un dato incorrecto.	

Paso 3: Mediante la utilización del tercer paso en el cual se realiza el diseño de la aplicación, se dividieron las HU en tareas y se procedió a realizar el diseño de los componentes y la aplicación en sí, además de la implementación del diseño. Quedando conformada con los componentes antes mencionados y con un diseño que agradó al cliente. Se generaron las planillas de especificación de los servicios consumidos y de los componentes, la lista de tareas y el archivo del diseño.

Tabla 2. Planilla de especificación del servicio Estado del Tiempo. Fuente: Elaboración propia.

Especificación de servicios que consume	
Nombre del servicio.	Estado del tiempo.
Propósito del servicio.	Brindar el estado del tiempo y la temperatura de cada día.
Dominio del servicio.	uci.cu.

Lenguaje del servicio	Java
Dependencias de otros servicios	No.
Entradas al servicio.	No.
Salidas del servicio.	String

Archivo de diseño. Servicio Listado de películas.

```
function getpelis () {
var mode=document.getElementById ("int").value;
var operation=services ["admin/prueba"].operations ["internos"];
var payload= operation.payloadJSON ();
payload ["p: internos"].objeto.$ = mode;
operation.callback = function (payload) {
  var responseXML = WSRequest.util._serializeToString  (payload);
  var responseJSON = WebService.utils.xml2bf (payload);
  var alter  = responseJSON ["ws: internosResponse"]["return"].$;
imprime (alter);
};
```

Figura 18. Archivo del servicio Listado de Películas. Fuente: Elaboración propia.

Paso 4: En la ejecución del cuarto paso se procedió a realizarle a la aplicación las pruebas de aceptación. Las cuales arrojaron que el servicio de Consumir el estado del tiempo no funcionaba correctamente, ya que no ofrecía todos los estados de tiempo. Así que se procedió a refactorizar el código de esta funcionalidad y a comprobar su funcionamiento nuevamente, pasando las pruebas con éxito la segunda vez. En este paso se generó el artefacto Lista de pruebas de aceptación pasadas exitosamente.

Tabla 3. Planilla de prueba de aceptación pasada exitosamente. Fuente: Elaboración propia.

Prueba de Aceptación	
Código: HU7-P1	**Nombre:** Insertar dato

Descripción: Se le introduce una variable de valor nulo a la aplicación, y se espera que muestre un mensaje de error por introducir un dato incorrecto.
Condiciones de ejecución: El usuario debe haber solicitado mandar un correo con las opciones recreativas
Entrada/Pasos de ejecución: 1- Se entra un usuario válido. 2- Se entra un usuario no válido.
Resultado esperado: 1- La aplicación muestra un mensaje indicando el éxito de la operación. 2- La aplicación muestra un mensaje indicando que el dato no es válido.
Resultado obtenido: 1- La aplicación muestra un mensaje indicando el éxito de la operación. 2- La aplicación muestra un mensaje indicando que el dato no es válido.
Evaluación de la prueba: Satisfactoria.

3.2.2 Descripción del caso de estudio Información Estudiante.

En la Universidad de las Ciencias Informáticas existen un número considerable de portales que contribuyen a mantener actualizada a toda la comunidad universitaria, además de posibilitar el intercambio de conocimientos. En el centro CDAE perteneciente a la facultad 5 de esta universidad, se necesita la incorporación de un sistema que permita gestionar de una manera sencilla la información, conocimientos y aplicaciones referentes al centro en las diferentes esferas, y mantenga informados y actualizados a sus miembros. El portal constará de una audiencia conformada por dos grupos temáticos, los estudiantes y profesores del CDAE; y un grupo más general pero no menos importante que lo constituye primeros grupos enmarcados en un ámbito interno y el último en un marco exterior a las necesidades comunicativas e informativas del centro.

3.2.2.1 Aplicación de la guía para el caso de estudio Información Estudiante.

Mediante la aplicación de la guía de trabajo al caso de estudio Información Estudiante se pudo constatar que el proceso de diseño de la aplicación compuesta mediante los pasos determinados en la guía brinda la efectividad requerida, teniendo en cuenta que para la realización anterior del caso de estudio también se utilizó la metodología XP y su proceso de desarrollo tomó 8 días. Mientras que con las modificaciones que se le realizaron a la metodología en la guía, el desarrollo del caso de estudio fue de 4 días. Además la utilización de la guía contribuyó a la obtención de un menor número de inconformidades producto de un diseño mejor estructurado que el anterior, se le realizaron con más facilidad las pruebas de aceptación a la aplicación; y se alcanzó un mayor grado de organización en la generación y utilización de artefactos. Ahora se explicarán los resultados obtenidos en la aplicación de cada paso y se mostrarán algunos artefactos generados a modo de ejemplo.

Paso 1: En el primer paso de la guía para el diseño de la aplicación se determinó que solo se iba a consumir un servicio y se iban a bridar dos:

Consume: Obtener Estudiante. Esta funcionalidad se basa en obtener un estudiante de la base de datos del portal Akademos.uci.cu, mediante la introducción del identificador del estudiante.
Brinda: Buscar Estudiante. Esta funcionalidad permite buscar un estudiante determinado a través de los criterios de búsqueda que introduzca el usuario.
Brinda: Modificar Estudiante. Esta funcionalidad permite modificar la información del estudiante que se haya buscado, permitiendo cambiarle uno o varios atributos al estudiante.

Se seleccionaron para ser utilizados en la interfaz gráfica los **componentes** siguientes: Tabla, Botón, Lista desplegable, Layets, Combobox y Embebed.

Y se generaron un diagrama de actividades y el siguiente **diagrama de secuencia:**

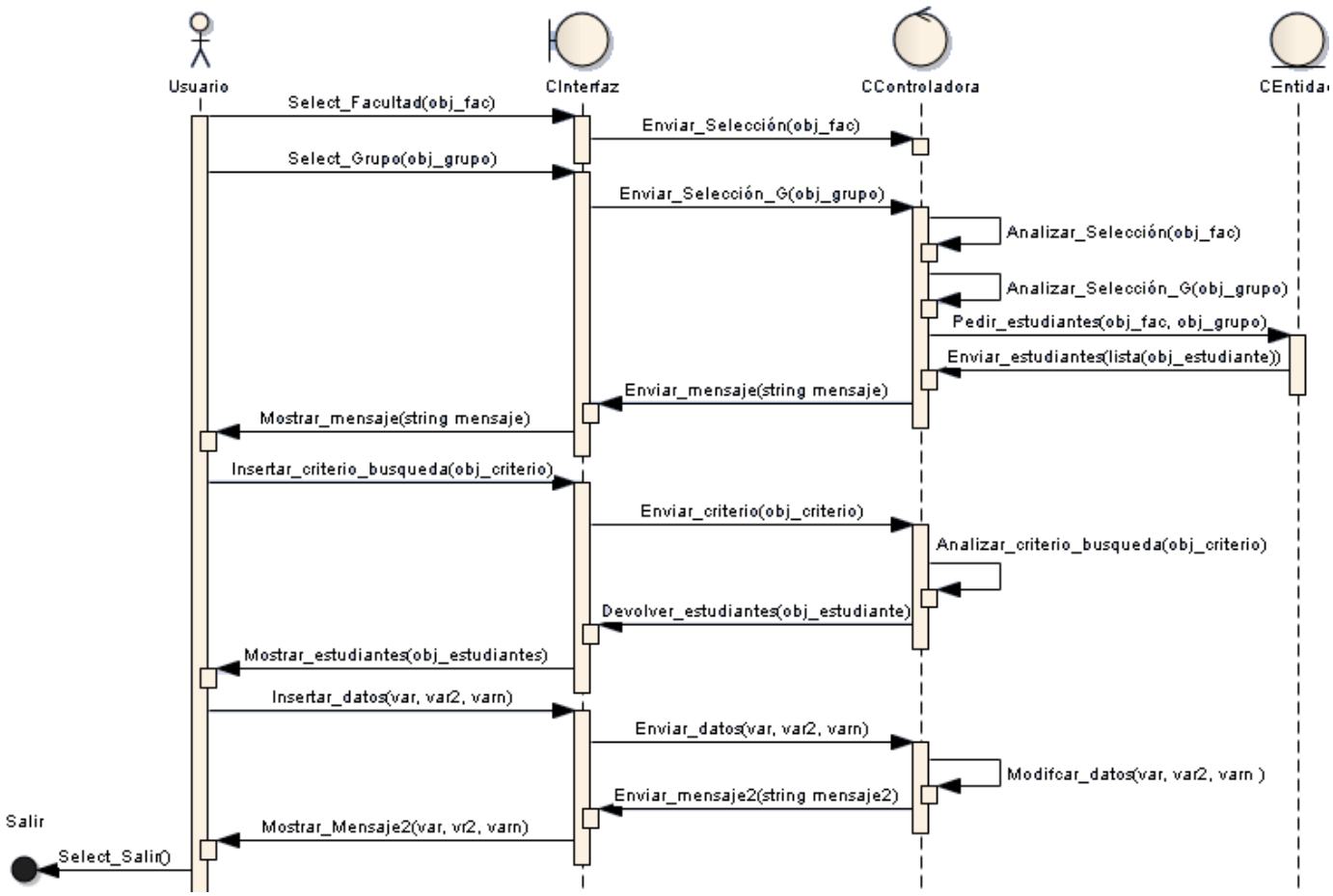

Figura 19. Diagrama de secuencia del caso de estudio Información Estudiante. Fuente: Elaboración propia.

Paso 2: En la utilización del segundo paso de la guía que está destinado a crear las pruebas para validar el funcionamiento de la aplicación. Se crearon las pruebas para comprobar que fuera posible importar, buscar y modificar un estudiante. No se generó ningún archivo de pruebas de aceptación, debido a la sencillez de las pruebas a realizar.

Tabla 4. Planilla de prueba de aceptación

# Prueba: 4	**HU # 1**

Nombre prueba: Prueba de la aplicación importar estudiante
Rol responsable: Programador: Dimelza Remedios Cruz.
Descripción: Se le realizan una serie de pruebas funcionales a la aplicación, con la finalidad de obtener posibles fallas.

Paso 3: El paso tres produjo el diseño de la aplicación, se dividieron las Historias de Usuario en tareas y se procedió a realizar el diseño de los componentes y la aplicación en sí, además de la implementación del diseño. Quedando conformada con los componentes antes mencionados y con un diseño que agradó al cliente. Se generaron las planillas de especificación de los servicios consumidos y de los componentes, la lista de tareas y el archivo del diseño. Para ejemplificar el desarrollo de este paso se muestran algunas de las tareas en las que se dividieron las Historias de Usuario.

Tabla 5. Plantilla de tarea 1 de la HU 1. Fuente: Elaboración propia.

# Tarea: 1	**HU # 1**
Nombre tarea: Diseño de la interfaz Importar estudiante.	
Tipo de tarea: Diseño	
Fecha inicio: 21/06/2012	**Fecha fin:** 22/06/2012
Programador responsable: Dimelza Remedios Cruz	
Descripción: Se realiza el diseño de la pantalla modificar profesor. Se crea el diseño realizando una correspondencia entre los tipos de datos a entrar, los datos soportados y los componentes visuales que más se adapten a estos.	

Tabla 6. Plantilla de tarea 1 de la HU 2. Fuente: Elaboración propia.

# Tarea: 1	**HU # 2**
Nombre tarea: Diseño de la pantalla para buscar estudiantes.	
Tipo de tarea: Diseño	
Fecha inicio: 23/06/2012	**Fecha fin:** 23/06/2012

Programador responsable: Dimelza Remedios Cruz
Descripción: Se realiza el diseño de la pantalla buscar estudiantes, la cual presenta los criterios por los cuales se puede filtrar un listado de estudiantes.

Tabla 7. Plantilla de tarea 1 de la HU 3. Fuente: Elaboración propia.

Número tarea: 1	**HU # 3**
Nombre tarea: Diseño de la interfaz modificar estudiante.	
Tipo de tarea: Diseño	
Fecha inicio: 23/06/2012	**Fecha fin:** 24/06/2012
Programador responsable: Dimelza Remedios Cruz	
Descripción: Se realiza el diseño de la pantalla modificar profesor. Se crea el diseño realizando una correspondencia entre los tipos de datos a entrar, los datos soportados y los componentes visuales que más se adapten a estos.	

Paso 4: En la ejecución del cuarto paso se procedió a realizarle a la aplicación las pruebas de aceptación. Las cuales arrojaron que todos los servicios funcionaban correctamente. En este paso se generó el artefacto Lista de pruebas de aceptación pasadas exitosamente.

Tabla 8. Planilla de prueba de aceptación pasada exitosamente. Fuente: Elaboración propia.

Prueba de Aceptación	
Código: HU1-P1	**Nombre:** Importar estudiantes
Descripción: Con esta prueba se validara que el registro de los estudiantes vinculados al centro CDAE en la aplicación se realiza de la manera esperada. Los datos asociados a los estudiantes deben ser importados mediante el consumo de servicios del Sistema de Gestión Académica de la Universidad (Akademos).	
Condiciones de ejecución: El usuario debe de estar autenticado y tener privilegios de administrador	

para tener acceso a este servicio.
Entrada/Pasos de ejecución: De un listado proporcionado por la aplicación, se selecciona primero la facultad y después el grupo al que pertenecen los estudiantes. Luego se pulsa el botón importar y los datos son almacenados en la base de datos del portal.
Resultado esperado: Mostar un mensaje indicando el éxito de la operación.
Resultado obtenido: Se muestra un mensaje indicando el éxito de la operación realizada.
Evaluación de la prueba: Satisfactoria.

3.3 Conclusiones parciales.

En el presente capítulo se realizó la validación de la propuesta de guía de trabajo presentada en el capítulo anterior. Para lo cual se utilizó el método matemático Delphi y la realización de varios casos de estudio donde se determinó que la funcionalidad de la guía de trabajo era la esperada. Mientras que a través del método Delphi se seleccionaron varios expertos, a los que se les presentaron dos cuestionarios; y después se realizó el análisis estadístico de los resultados del último cuestionario presentado. Para concluir con un proceso de validación donde el 90% de los expertos consideran que la propuesta de guía de trabajo sobre las prácticas Diseño Incremental y Pruebas Continuas para el desarrollo de aplicaciones compuestas tiene la calidad requerida.

Conclusiones Generales

Tras la realización del presente trabajo, se ha podido arribar a las siguientes conclusiones:

Primeramente se realizó un marco teórico, que permitió conocer el estado actual de la metodología de desarrollo de software XP y su vinculación con el desarrollo de aplicaciones compuestas. Además de la importancia que tienen las prácticas Diseño Incremental y Pruebas Continuas dentro de su proceso de desarrollo.

La realización de este marco permitió llegar al capítulo dos con suficiente información sobre la metodología ágil XP. Lo que contribuyó a la creación de la guía de trabajo sobre las prácticas Diseño Incremental y Pruebas Continuas de esta metodología para el desarrollo de aplicaciones compuestas. Además de ayudar a determinar la necesidad de modificar el proceso de desarrollo que plantea la metodología para estas dos prácticas, en función del desarrollo específico de aplicaciones compuestas.

Por último en el capítulo tres se validó la guía de trabajo, mediante la utilización del Método Delphi tras lo cual la misma fue calificada de válida por el 90% de los expertos consultados. Más la realización de varios casos de estudio, donde se aplicó la guía para crear el diseño de aplicaciones compuestas. Concluyendo que la guía agiliza y le da un mayor grado de organización al proceso de diseño.

Recomendaciones

- Sugerir que se realice un estudio de los frameworks y herramientas que se utilizan para el desarrollo de aplicaciones compuestas con vistas a incluirlos en el proceso que plantea la guía de trabajo propuesta.

- Realizar un estudio de la etapa de producción de la metodología XP y de las aplicaciones compuestas, con vistas a extender la guía de trabajo a esta etapa.

Referencias Bibliográficas

1. Castillo, J.L.D., Factores determinntes y críticos en empresas de servicios, para la obtencion de ventajas competitivas sostenibles y transferibles a estraegias de globalizacion: Un análisis de la industria del software. 2004, Universidad Autonoma de Barcelona Barcelona. p. 548.

2. Thuraisingham, B., Secure Semantic Service-Oriented Systems. 2011.

3. Banerjee, A. (2006) What Are Composite Applications?

4. Glenn, Prisma Patrones para crear aplicaciones compuestas con WPF, in MSDN. 2011.

5. Corporation, I. Solución Informatica MDM. 2011.

6. Kent Beck, C.A., Extreme Programming Explained: Embrace Change. 2004.

7. Gómez, W.J.A. (2007) Metodología de desarrollo de software un enfoque práctico y global versión 1.0.11 beta (20071020). 15.

8. Ramiro, C.S., Metodología de Gestión de Proyectos en las Admistraciones Públicas según ISO 10.006, in Departmento de explotación y prospección de minas. 2007, Universidad de Oviedo: Oviedo. p. 312.

9. Reynoso, C. (2004) De los métodos heterodoxos en la construcción de software.

10. Marchesi, M. (2005) The New XP.

11. Sopeña, J.G., F.J.S. Camino, and J.J.M.N. (coordinador) (2008) tecnologías software orientadas a servicios.

12. Franco Ordoñez Leon, R.R.S. (2011) Desarrollo de un mashup comercial para la publicación y búsqueda de bienes inmuebles en quito integrando distintas apis públicas y proveyendo de las interfaces necesarias para que sea consumido desde cualquier tipo de aplicación informática.

13. Tom Deutsch, K.M. New Approaches to Information Access and Assembly.

14. Mora, M.E.d.l., Metodología de la investigación: desarrollo de la inteligencia. 2006.

15. Cuaresma, M.J.E., Metodologías para el desarrollo de sistemas de información global: análisis comparativo y propuesta, in Departamento de Lenguajes y Sistemas Informáticos. 2001: Universidad de Sevilla.

16. Suárez, R.C., Metodología de gestión de proyectos en las administraciones públicas según ISO 10.006, in Departamento de explotación y prospección de minas. . 2007, Universidad de Oviedo.

17. Letelier Patricio, P.M.C. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). 17.

18. Stephens, M. and D. Rosenberg, Extreme Programming Refactored: The Case Against XP. 2003.
19. Beck, K., Extreme Programming Explained. 1999.
20. Wake, W.C., Extreme Programming Explored. Vol. 1. 2000.
21. Jeffries, R., A. Anderson, and C. Hendrickson, Extreme Programming Installed. 2000.
22. Lisa Crispin, T.H., Testing Extreme Programming. Vol. 1. 2002. 336.
23. Keyser, C., Aplicaciones Compuestas- el nuevo paradigma. The Architecture Journal, 2005. **10**.
24. Information Technology for European Advancement. Available from: http://www.itea-office.org.
25. Designer, A.d.L.D. Proceso de creación de aplicaciones compuestas Available from: http://lgpx09.lg.ehu.es/help/help85_designer.nsf/b3266a3c17f9bb7085256b870069c0a9/1f7e555740deae02c1257627006251ee?OpenDocument.
26. IBM, C. Creación de componentes NSF Available from: www.51869e04c0487b78c12576270062525f.htm.
27. IBM, C. Directrices básicas de diseño de componentes Available from: www.1391f4a464dd875ec125762700625249.htm.
28. Ramot, J.L. (2008) Mashup, sumando en la red.
29. EAFIT, U., Elementos de la Ingeniería de Software.
30. Yee, R., Pro Web 2.0 Mashups Remixing Data and Web Services. 2008.
31. Pérez, A.J.M. and A.L. Mojarena, Propuesta de Estrategia de Pruebas para los Software del Centro de Telemática. 2010: C. de La Habana.
32. Cristóbal Fransi, E.G.A., María Jesús, Desarrollo del Comercio Electrónico en la Gestión Empresarial. Análisis de su situación en España. 2007.
33. Astigarraga, E. El método Delphi.
34. Torres, C.F., Diseño de una estrategia de monitoreo y control de servicios en el proyecto SIGES-DT. 2010.
35. López., Y.V., Estrategia para el diseño de niveles de servicios utilizando la metodología ITIL en un entorno de producción de software en la Universidad de las Ciencias Informáticas. 2010.
36. Pavó, D.T., Propuesta de herramienta para el desarrollo de aplicaciones híbridas. 2012, Universidad de las Ciencias Informáticas: Ciudad de la Habana.

Glosario de Términos y Siglas

Actividad: Conjunto de acciones planificadas llevadas a cabo por una o más personas, tienen como finalidad alcanzar los objetivos trazados por la organización.

Competencias: Conjunto de atributos que una persona posee y le permiten desarrollar acción efectiva en determinado ámbito.

Repositorio: Sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

Proceso: Conjunto de actividades o eventos que se realizan o suceden con un determinado fin. Cada proceso tiene entradas, funciones y salidas.

Refactorización: Se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

Blog: Se trata de sitios Web donde se recopilan cronológicamente mensajes de uno o varios autores sobre una determinada temática a modo de diario personal.

Anexos

Anexo 1. Historia de Usuario propuesta en la Guía de trabajo para ser usada en el desarrollo de aplicaciones compuestas.

Historia de Usuario	
Número:	**Nombre Historia de Usuario:**
Modificación (o extensión) de Historia de Usuario (Nro. Y Nombre):	
Usuario:	**Iteración Asignada:**
Prioridad en Negocio: (Alta / Media / Baja)	**Puntos estimados**
Riesgo en Desarrollo: (Alto / Medio / Bajo)	**Puntos reales**
Descripción:	
Observaciones:	

Anexo 2. Tabla resumen sobre las diferencias entre metodologías ágiles y no ágiles.

Tabla 9. Diferencias entre metodologías ágiles y no ágiles. Fuente: Artículo “Metodologías ágiles para el desarrollo de software: Extreme Programming (XP)”.

Metodologías Ágiles	**Metodologías Tradicionales**

Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo de desarrollo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas.
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Anexo 3. Tabla del Coeficiente de argumentación.

Fuentes de Argumentación	**Grado de Influencia de Cada Fuente**		
	Alto	**Medio**	**Bajo**

Conocimiento propio del estado del problema en el extranjero	0.5	0.5	0.5
Análisis teóricos realizados por usted.	0.3	0.2	0.1
Su intuición.	0.5	0.5	0.5
Experiencia obtenida con el tema de la investigación.	0.5	0.4	0.2
Trabajo de autores nacionales.	0.5	0.5	0.5
Trabajo de autores extranjeros.	0.5	0.5	0.5
Totales	**1.0**	**0.8**	**0.5**

Anexo 4. Encuesta para obtener el coeficiente de conocimiento de los posibles expertos.

Estimado compañero (a) usted ha sido seleccionado (a) como posible experto por su experiencia y calificación profesional, con el objetivo de valorar la pertinencia y calidad del aporte que se propone en este trabajo investigativo, por ello se le solicita su cooperación. La aplicación del criterio de expertos requiere de algunos datos, los cuales se recopilarán a través de las siguientes preguntas, antes de someter a consulta la propuesta del autor.

1. Datos generales.

Nombre y Apellidos: _____

Título universitario: _____ Categoría docente: _____

Grado académico y/o científico._____ Centro de trabajo: _____

Cargo que desempeña: _____

2. Marque con una cruz en la siguiente tabla, el valor que considere se corresponda con el grado de conocimiento que posee sobre el desarrollo de aplicaciones compuestas. Teniendo en cuenta que 10 corresponde a un nivel elevado de conocimiento, y 0 corresponde a no conocer nada.

1	2	3	4	5	6	7	8	9	10

3. Valore el grado de influencia que han tenido diversas fuentes, en el nivel de conocimiento que posee sobre el desarrollo de aplicaciones compuestas. Marque con una cruz el grado de influencia en Alto (A), Medio (M) y Bajo (B) de cada una de las fuentes reflejadas en la siguiente tabla:

Fuentes de Argumentación	**Grado de Influencia de Cada Fuente**		
	Alto (A)	**Medio (M)**	**Bajo (B)**
Análisis teóricos realizados por usted.			
Experiencia obtenida con el tema de la investigación.			
Trabajo de otros autores.			
Su propio conocimiento del estado del problema en el extranjero.			
Su intuición.			

4. Marque con una cruz en la siguiente tabla, el valor que considere se corresponda con el grado de conocimiento que posee sobre proceso de desarrollo que plantea la metodología Extreme Programming. Teniendo en cuenta que 10 corresponde a un nivel elevado de conocimiento, y 0 corresponde a no conocer nada.

1	2	3	4	5	6	7	8	9	10

5. Valore el grado de influencia que han tenido diversas fuentes, en el nivel de conocimiento que posee sobre el desarrollo de aplicaciones compuestas mediante la metodología Extreme Programming. Marque con una cruz el grado de influencia en Alto (A), Medio (M) y Bajo (B) de cada una de las fuentes reflejadas en la siguiente tabla:

Fuentes de Argumentación	**Grado de Influencia de Cada Fuente**		
	Alto (A)	**Medio (M)**	**Bajo (B)**
Análisis teóricos realizados por usted.			
Experiencia obtenida con el tema de la investigación.			
Trabajo de otros autores.			
Su propio conocimiento del estado del problema en el extranjero.			
Su intuición.			

Anexo 5. Encuesta para validar la guía de trabajo propuesta.

Compañero (a): El presente trabajo de diploma se propone entre sus objetivos la validación de la estrategia para el Desarrollo de Aplicaciones Compuestas. Con este fin se solicita su valiosa colaboración para evaluar la guía de trabajo propuesta. Por lo que deseamos que usted responda a las preguntas que se encuentran a continuación:

1. ¿Es importante la implantación de una guía de trabajo que ayude a desarrollar aplicaciones compuestas?

Sí ____ No____

2. ¿Cree usted que la metodología escogida fue la más indicada para el desarrollo de una guía de trabajo para la realización de aplicaciones compuestas?

Sí ____ No____

3. ¿Cree usted que las prácticas escogidas fueron las más indicadas para el desarrollo de una guía de trabajo sobre la realización de aplicaciones compuestas?

Sí ____ No____

4. ¿Considera usted que existe claridad y lógica en la definición de las etapas y pasos de la guía de trabajo?

Sí ____ No____

5. ¿Considera usted que las actividades descritas en la guía de trabajo cumple con las pautas fundamentales para un desarrollo exitoso de aplicaciones compuestas?

Sí ____ No____

6. ¿Considera usted que los roles incluidos en la guía de trabajo son los adecuados para el desarrollo de aplicaciones compuestas?

Sí ____ No____

7. ¿Considera usted que el grado de correspondencia que se establece entre los artefactos generados y las actividades planteadas es satisfactorio para el desarrollo de aplicaciones compuestas?

Sí ____ No____

Otras consideraciones que se tuvieron en cuenta para tomar la valoración de los encuestados son:

1. Evaluación personal del encuestado sobre la guía de trabajo: En este punto el encuestado emite una breve evaluación personal sobre la guía.

2. Fallas o problemas hallados por el encuestado en la guía de trabajo: En este punto el encuestado manifiesta los problemas o deficiencias hallados en la guía.

3. Recomendaciones del encuestado a la guía de trabajo: En este punto el encuestado expresa algunas recomendaciones para la guía.